Graphics and Imaging Overview

Session 200

Graphics and Imaging Overview

Travis Brown
Graphics and Imaging Evangelist

Graphics and Imaging Directions

Peter Graffagnino
Director, Graphics and Imaging Software

Agenda

- Graphics Architecture of Mac OS X
- PDF: Mac OS X's Digital Paper
- The GPU Computing Revolution
- What's New for Panther

Aqua

Frameworks

Graphics

Darwin

Aqua

Frameworks

Quartz

OpenGL

QuickTime

Quartz Compositor

Darwin

Aqua

Frameworks

Quartz

OpenGL

QuickTime

Quartz Compositor

Darwin

Aqua

Frameworks

Quartz

OpenGL

QuickTime

Quartz Compositor

Darwin

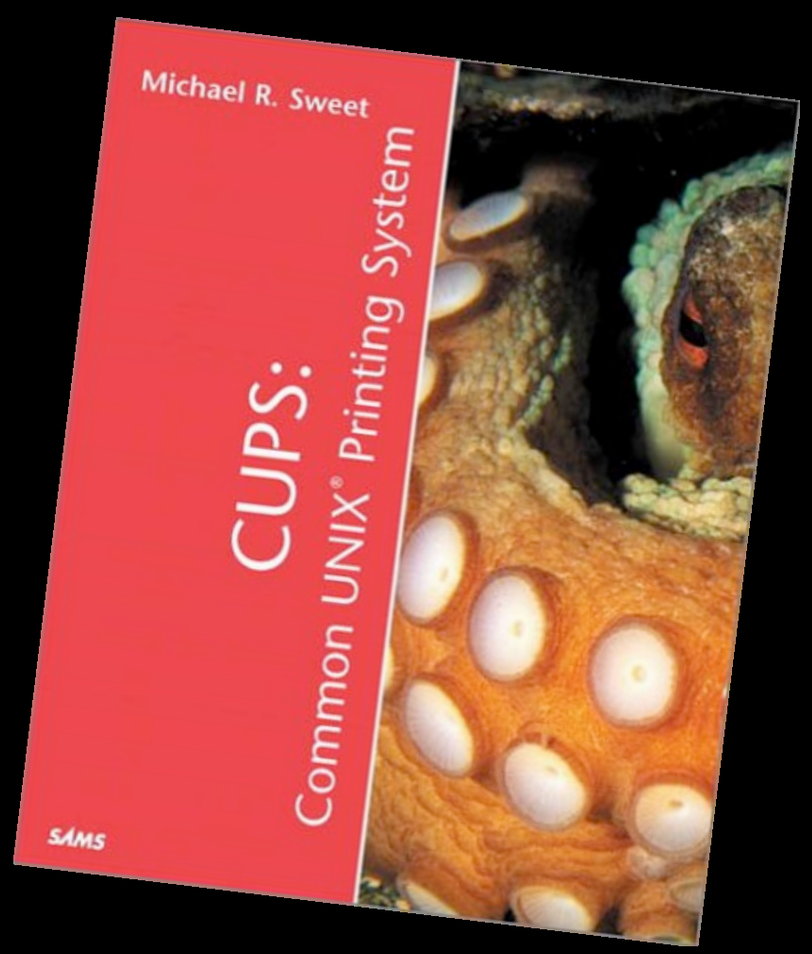
Industry Standard 2D Imaging Model

Quartz 2D

- Lightweight C Library
- Read/Write PDF
- Fast Anti-aliasing
- Destination Alpha
- Apple Type Services built-in
- Type 1 Scaler built-in
- ColorSync built-in

Printing With Quartz

- Same imaging model for Screen and Printing
- Spool file is just PDF
- High-quality Quartz rasterization for inkjets
- End-to-end color management
- Open Source infrastructure



Aqua

Frameworks

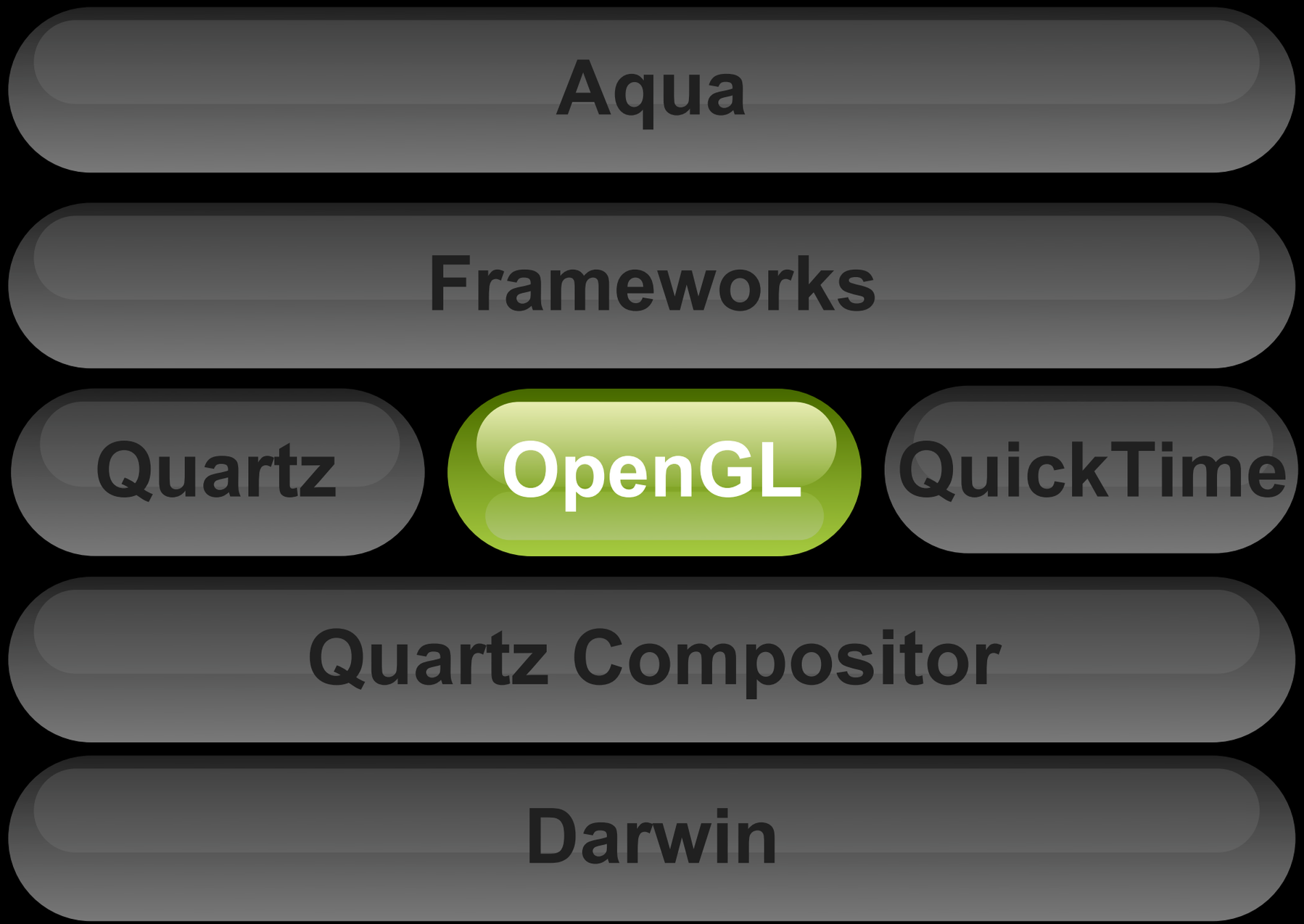
Quartz

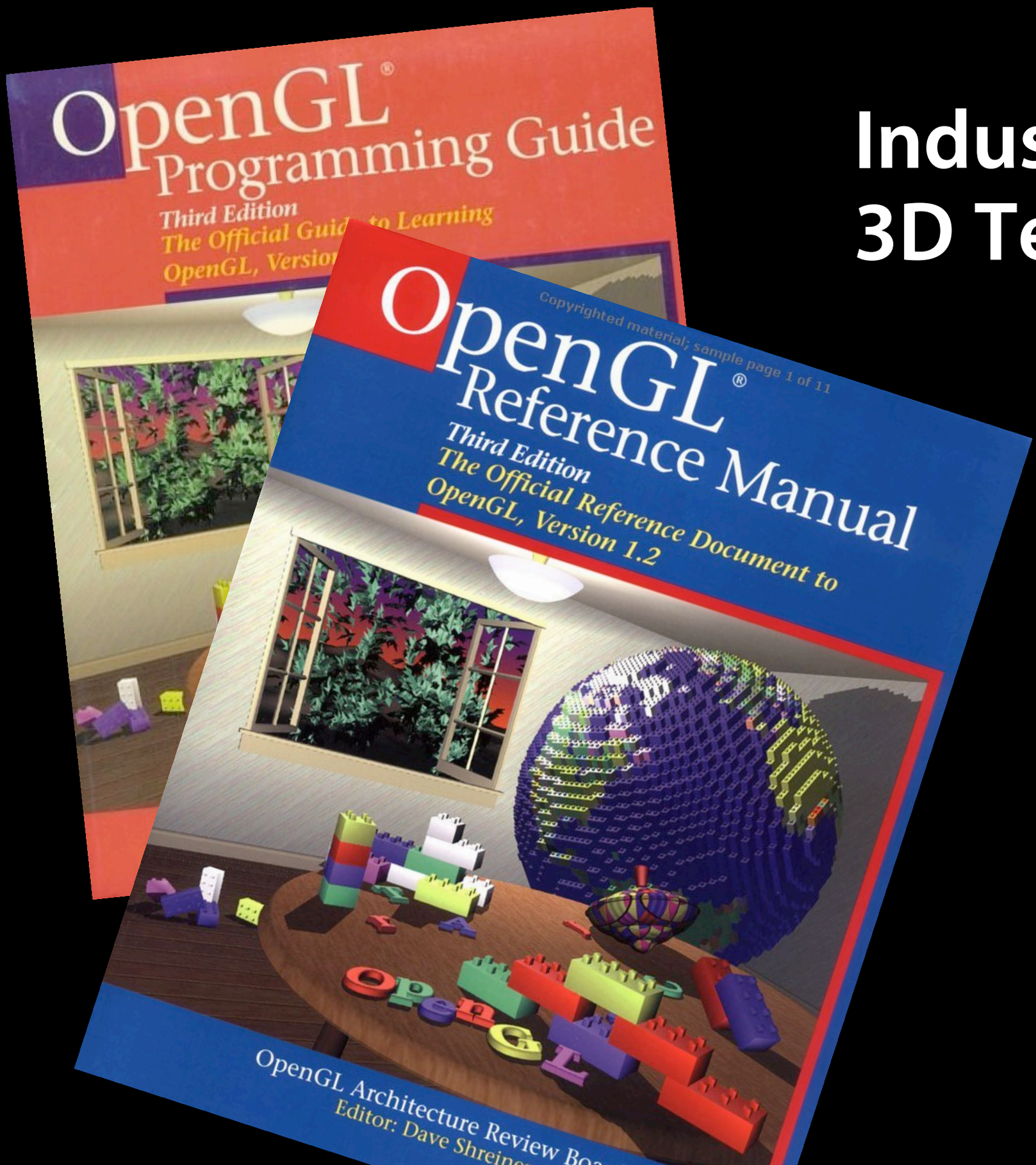
OpenGL

QuickTime

Quartz Compositor

Darwin


Aqua
Frameworks
Quartz
OpenGL
QuickTime
Quartz Compositor
Darwin



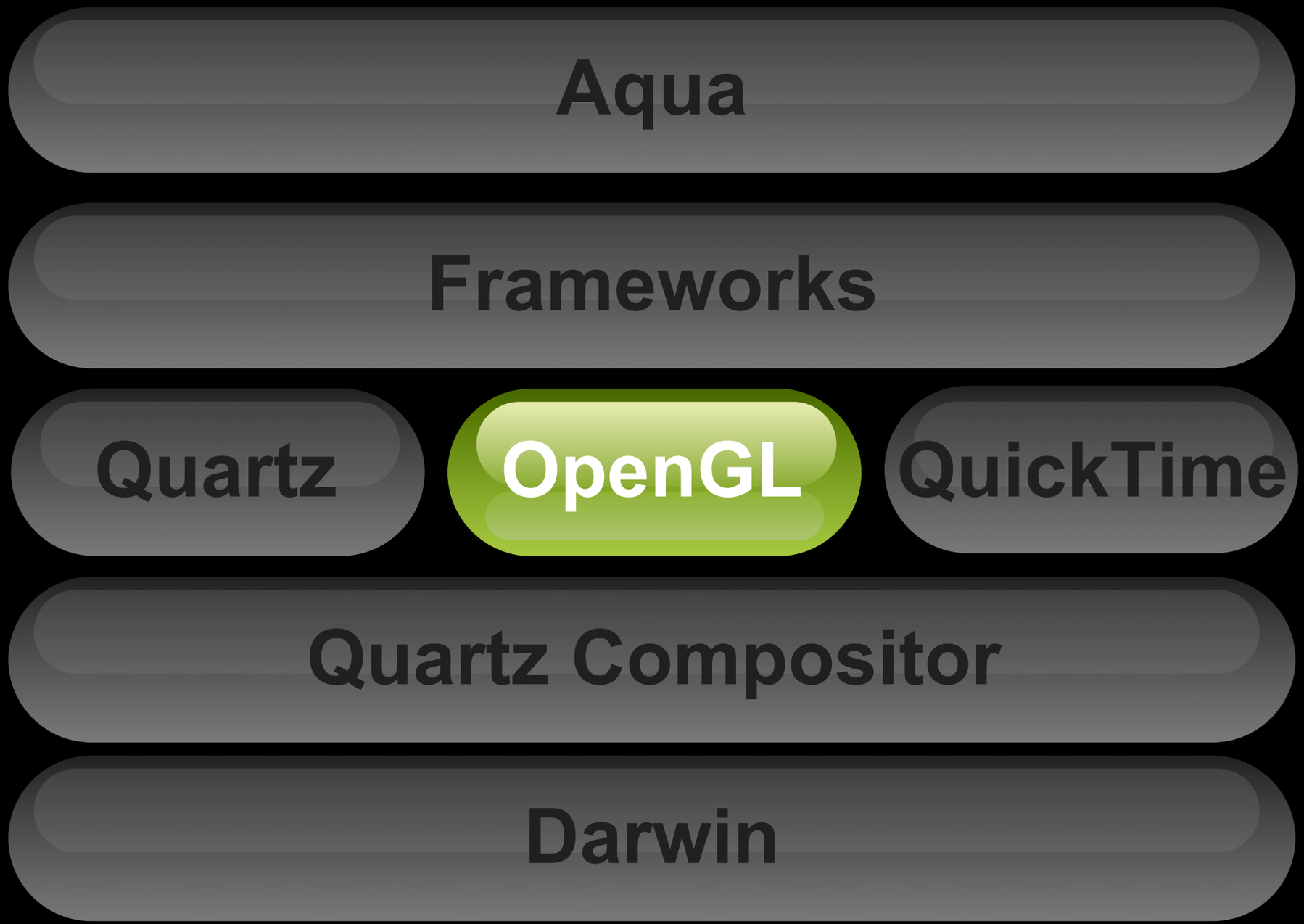
Industry Standard 3D Technology

Mac OS X OpenGL

- State-of-the-art architecture
  - Resource virtualization
  - Data flow optimizations
- Unified Driver Model
- Stability and Robustness
- Not just for games!

Mac OS X OpenGL

- Innovative uses of OpenGL in the OS
  - Quartz Extreme
- Innovative uses of OpenGL in consumer applications
  - Keynote, iChat
- Your app here next year!


Aqua
Frameworks
Quartz
OpenGL
QuickTime
Quartz Compositor
Darwin

Aqua

Frameworks

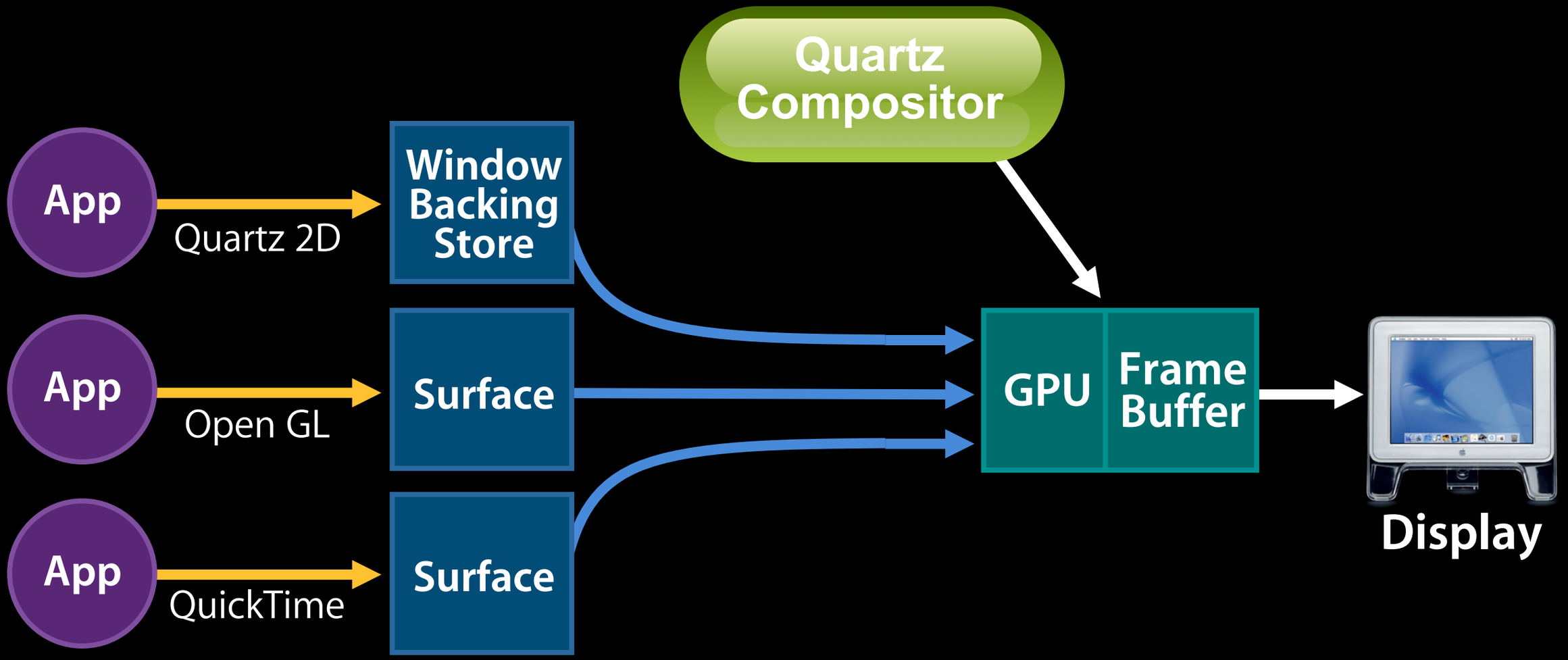
Quartz | OpenGL | QuickTime

Quartz Compositor

Darwin

Window System as a Digital Image Compositor

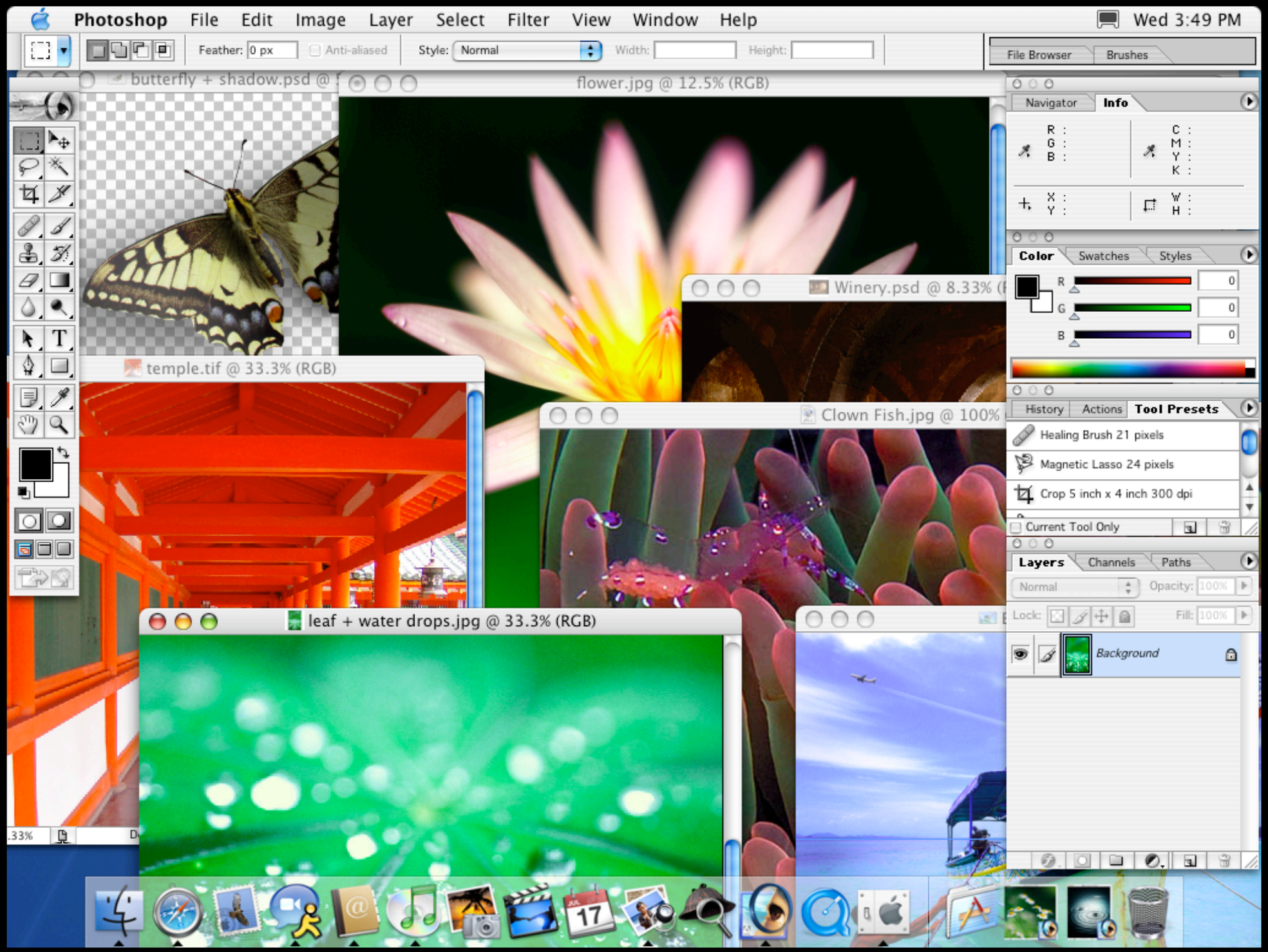
Quartz Extreme



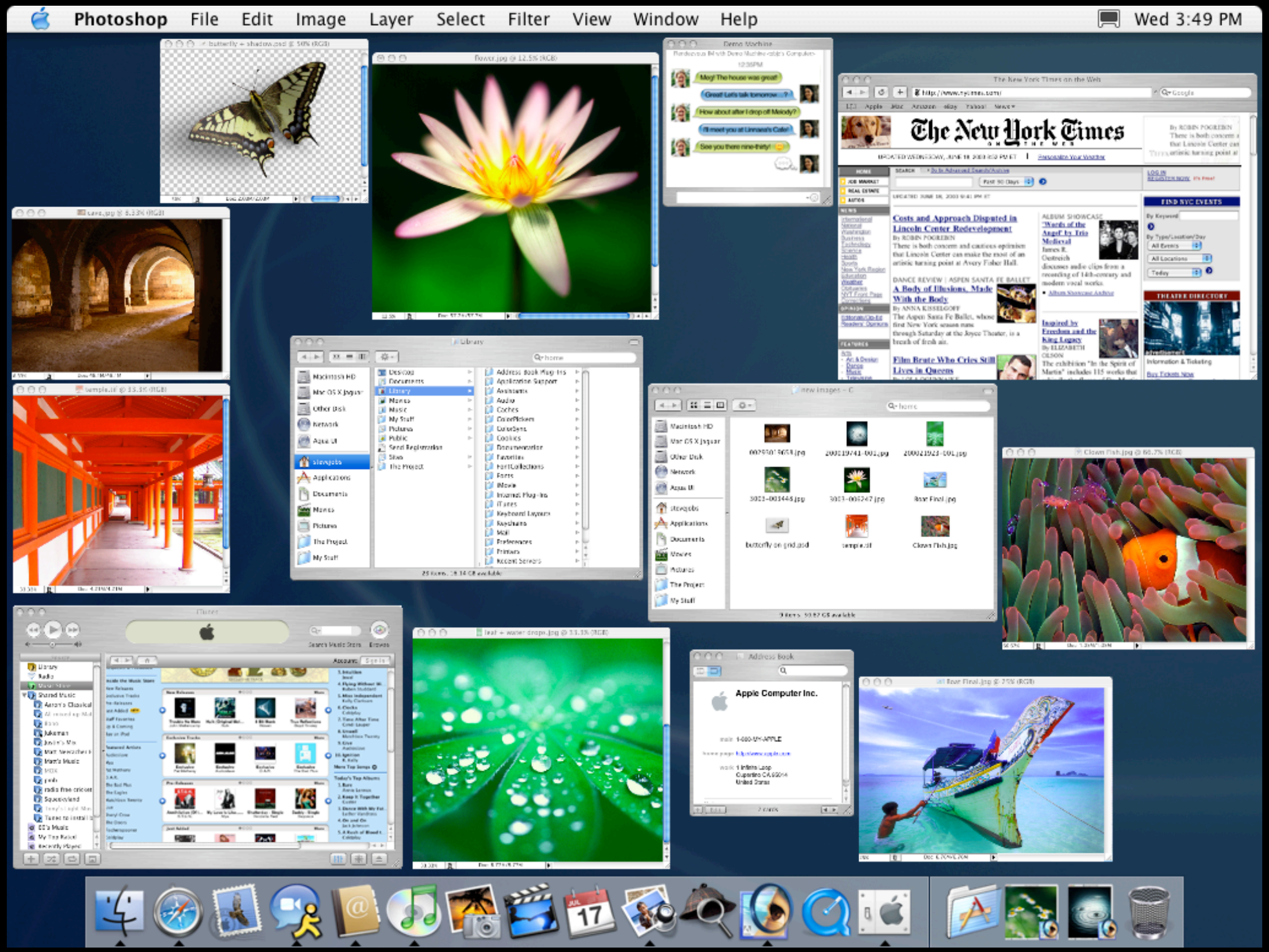
Apple Leading the Industry

- Panther is our 3rd generation desktop composition engine
- Natural evolution of windowing systems
- Apple OpenGL is robust enough for 24x7 operation
- Our advances are directly useable by you via OpenGL

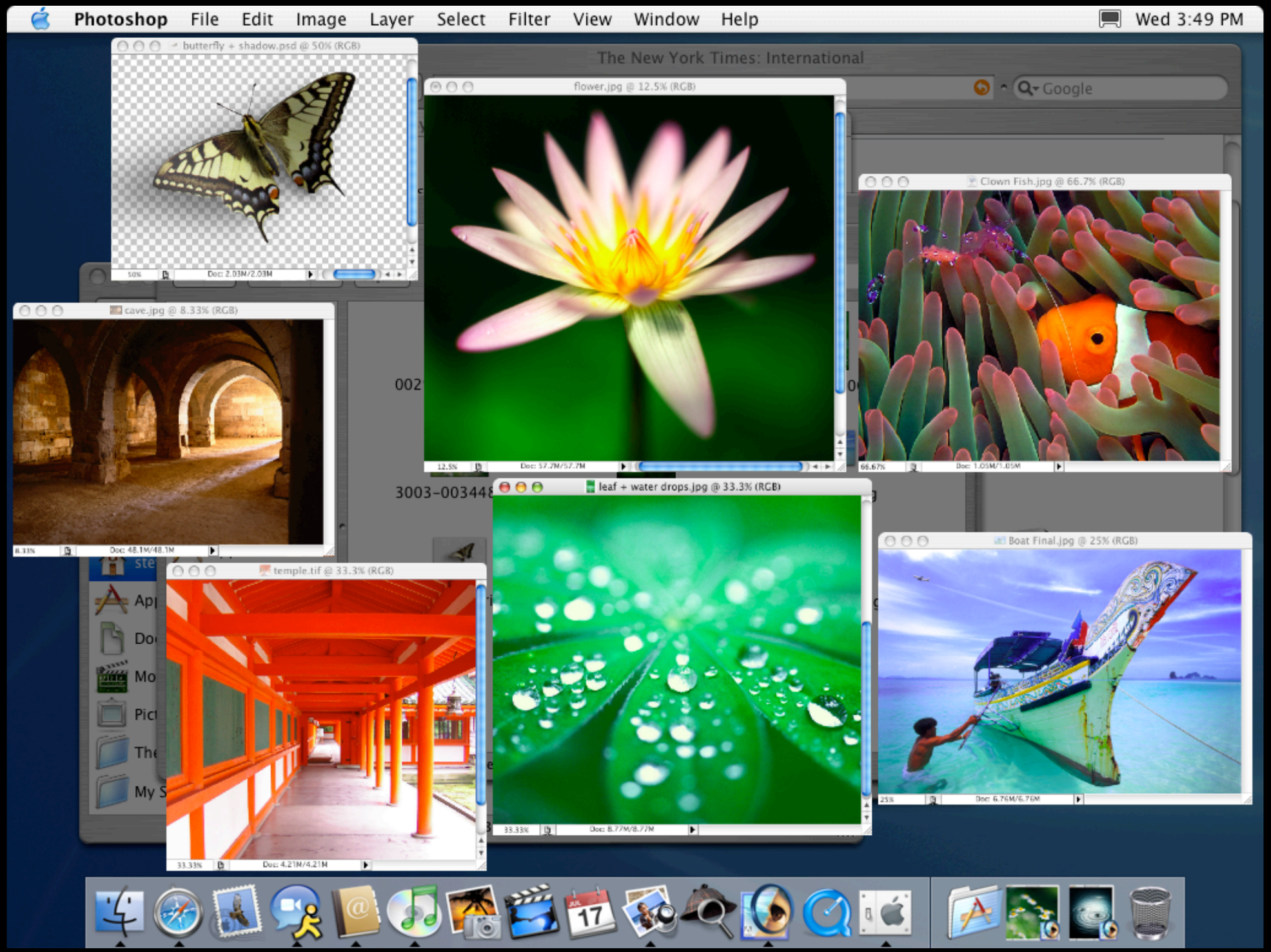
Exposé


Photoshop File Edit Image Layer Select Filter View Window Help
Wed 3:49 PM
Feather: 0 px
Anti-aliased
Style: Normal
Width:
Height:
File Browser
Brushes
butterfly + shadow.psd @
flower.jpg @ 12.5% (RGB)
Navigator
Info
R :
G :
B :
C :
M :
Y :
K :
X :
Y :
W :
H :
Color
Swatches
Styles
R
G
B
Winery.psd @ 8.33%
temple.tif @ 33.3% (RGB)
Clown Fish.jpg @ 100%
History
Actions
Tool Presets
Healing Brush 21 pixels
Magnetic Lasso 24 pixels
Crop 5 inch x 4 inch 300 dpi
Current Tool Only
Layers
Channels
Paths
Normal
Opacity:
Lock:
Fill:
Background
leaf + water drops.jpg @ 33.3% (RGB)

Exposé


Photoshop File Edit Image Layer Select Filter View Window Help
Wed 3:49 PM
The New York Times
Costs and Approach Disputed in Lincoln Center Redevelopment
A Body of Illusions, Made With the Body
Film Brute Who Cries Still Lives in Queens
Apple Computer Inc.

Exposé


Photoshop File Edit Image Layer Select Filter View Window Help
Wed 3:49 PM
butterfly + shadow.psd @ 50% (RGB)
The New York Times: International
Google
flower.jpg @ 12.5% (RGB)
Clown Fish.jpg @ 66.7% (RGB)
cave.jpg @ 8.33% (RGB)
leaf + water drops.jpg @ 33.3% (RGB)
Boat Final.jpg @ 25% (RGB)
temple.tif @ 33.3% (RGB)

Aqua

Frameworks

Quartz | OpenGL | QuickTime

Quartz Compositor

Darwin

Aqua

Frameworks

Quartz

OpenGL

QuickTime

Quartz Compositor

Darwin

PDF: Mac OS X's Digital Paper

Digital Paper

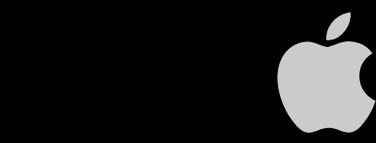
- PDF is a digital representation of the printed page
  - Application independent
  - Device independent
  - Extremely high fidelity
- 500 million viewers distributed
- Universal “view-level” abstraction in the MVC paradigm

MVC Paradigm

Controller

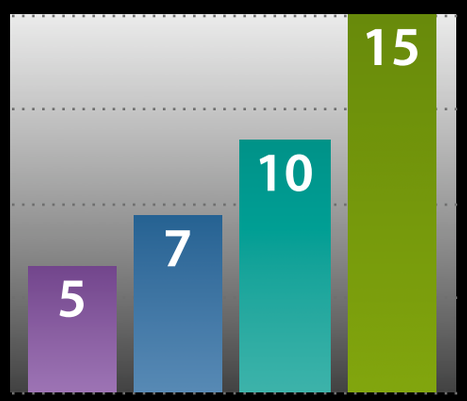
Model

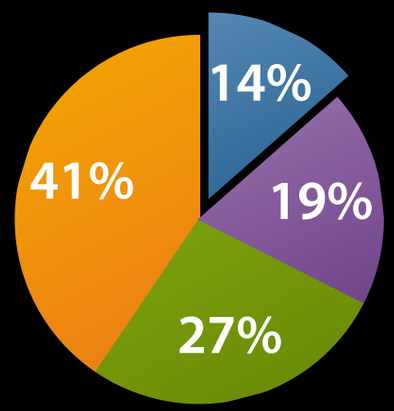
View

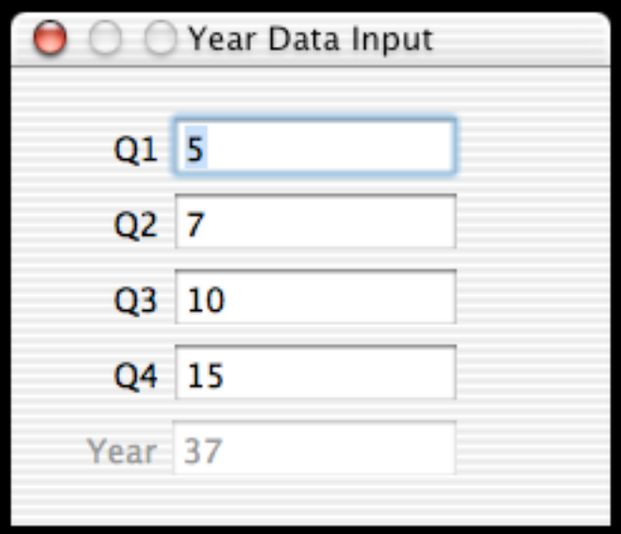


MVC Example

```
Q1 = 5,  Q2 = 7, Q3 = 10, Q4 = 15
Year = Q1 + Q2 + Q3 + Q4
```





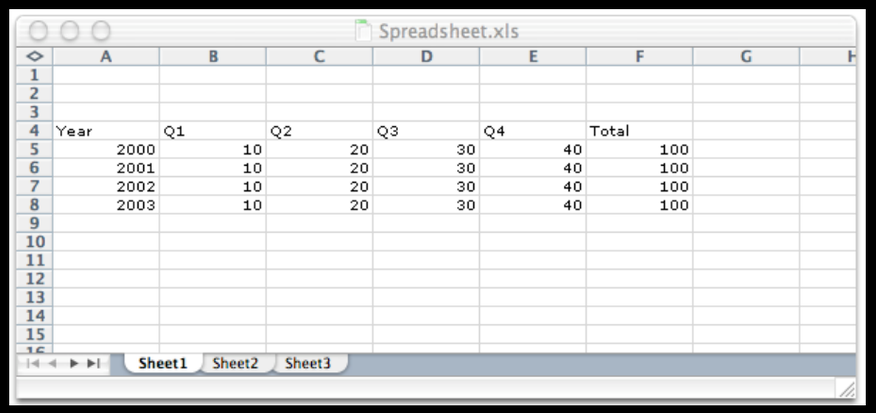


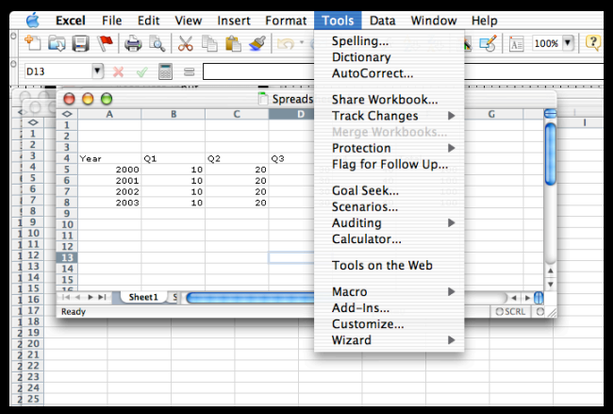
- Model: Application data structures
- View: Visual representation of model
- Controller: Manipulates model

The Value of MVC

- Models can have multiple views
- Models can have multiple controllers
- Any model can be “projected” onto a 2D representation
- Views of vastly different models can share a common visual language

Document-based Applications Are MVC





- Model: Application data structures, usually made persistent in document

- View: Visual representation of model in window

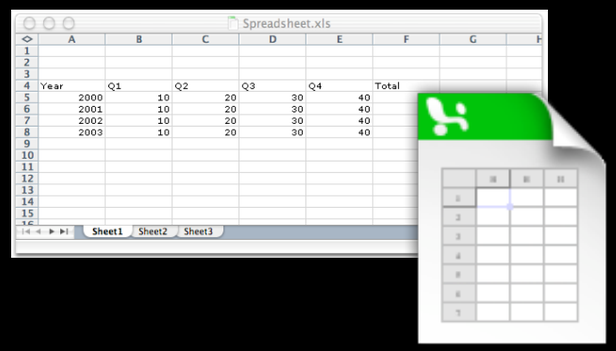
- Controller: User Interface of application

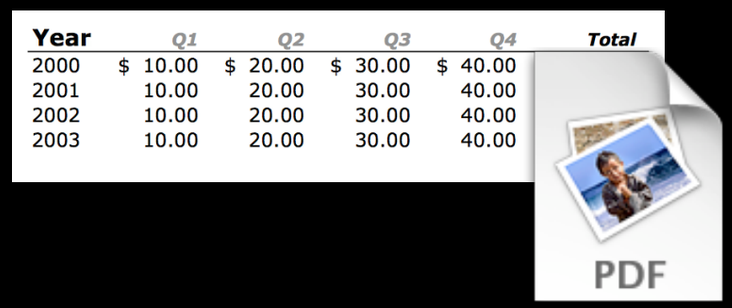
This Is Not a Spreadsheet

Year	Q1	Q2	Q3	Q4	Total
2000	$ 10.00	$ 20.00	$ 30.00	$ 40.00	$ 100.00
2001	10.00	20.00	30.00	40.00	100.00
2002	10.00	20.00	30.00	40.00	100.00
2003	10.00	20.00	30.00	40.00	100

...it's a *representation* of a spreadsheet

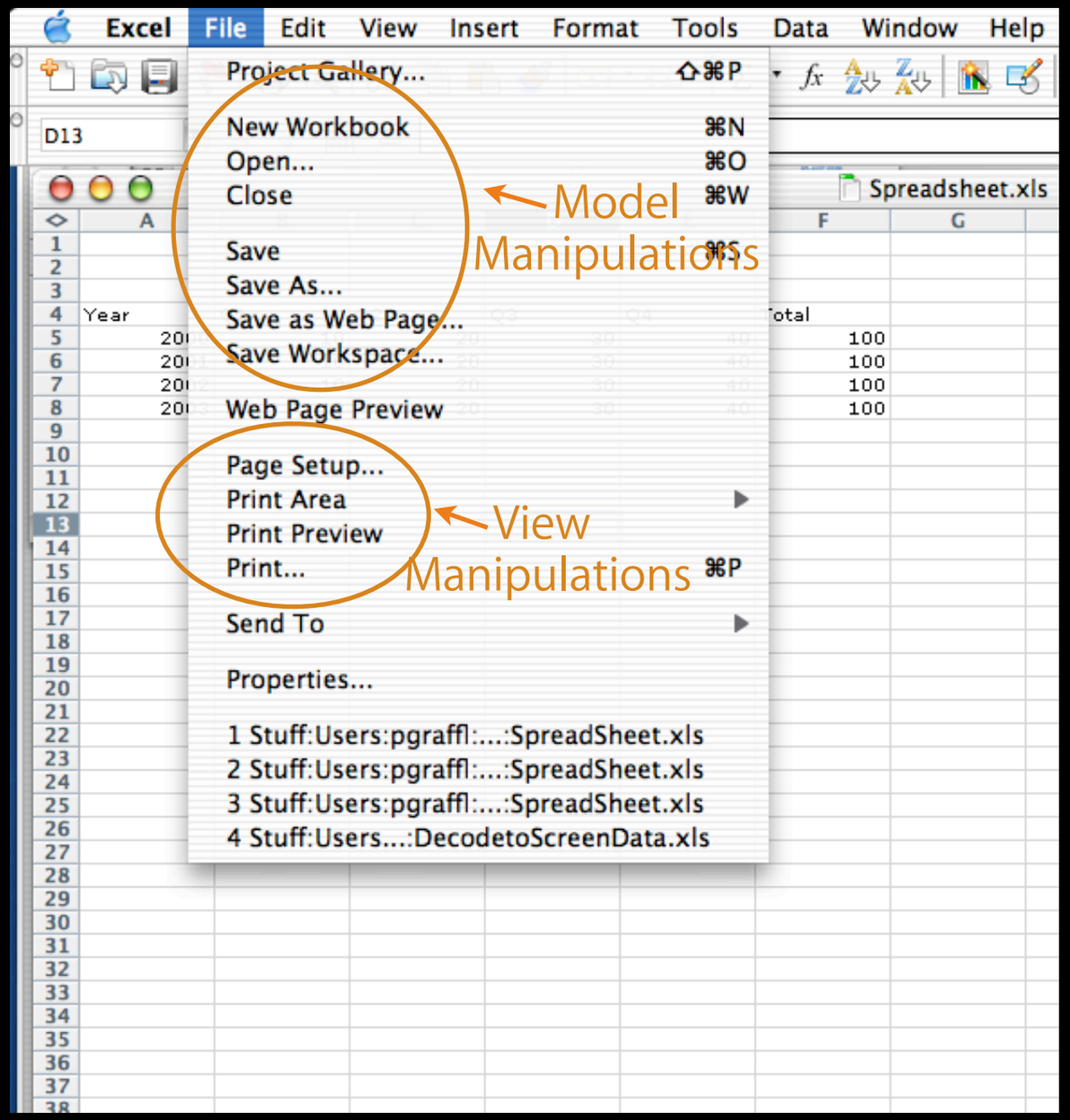
Model vs. View





- Hi-fidelity model representation
- Requires Application
- Configuration dependent
  - Fonts, plugins, app version

- Hi-fidelity view representation
- Does not require Application
- Configuration independent



Why Is "Save As PDF…" in the Print Panel?

- PDF is not a model level concept
- "Saving to PDF" is really just "Printing to Digital Paper"
- Mac OS X is the first commercial OS to have a system-wide standard for Digital Paper

PDF Is Not a Model Level Construct…

- Consider encoding editable, flowable text in PDF
- Imposes a model on a view level construct
- Text flow models are complicated
- A precise specification includes:
  - Styling model
  - Obstacle avoidance model
  - Pagination model (widows and orphans)
  - Justification model
  - Hyphenation model (requires language dictionary)

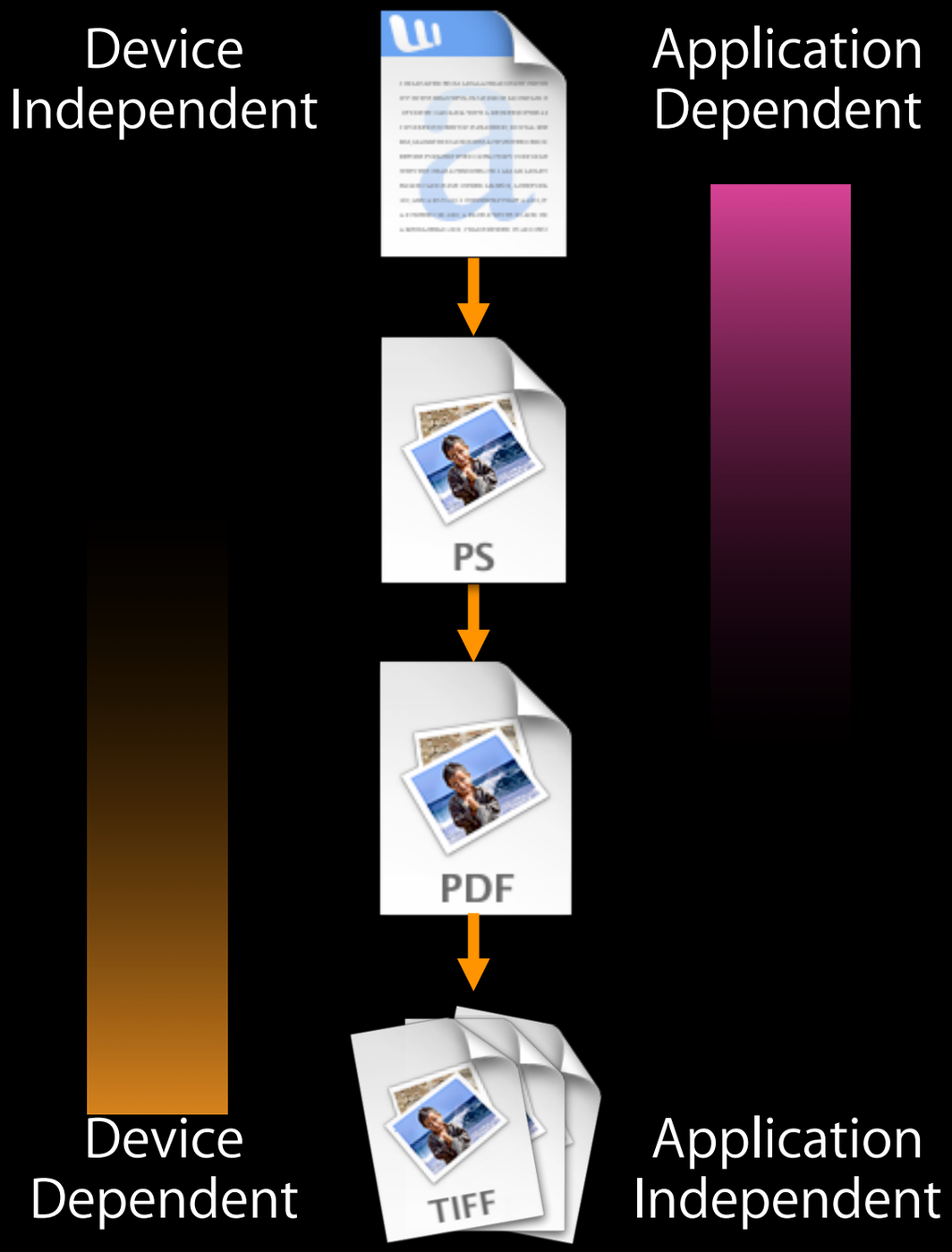
PDF Is Not a Good Fit for Carrying Model-level Data

- Sacrifices universality
  - PDF is a universal abstraction for marks on a page
  - There isn't a single universal abstraction for "flowable text documents", or "spreadsheets", nor should there be
- Models are precisely where applications innovate and differentiate
- We want apps to innovate and differentiate

Our Strategy

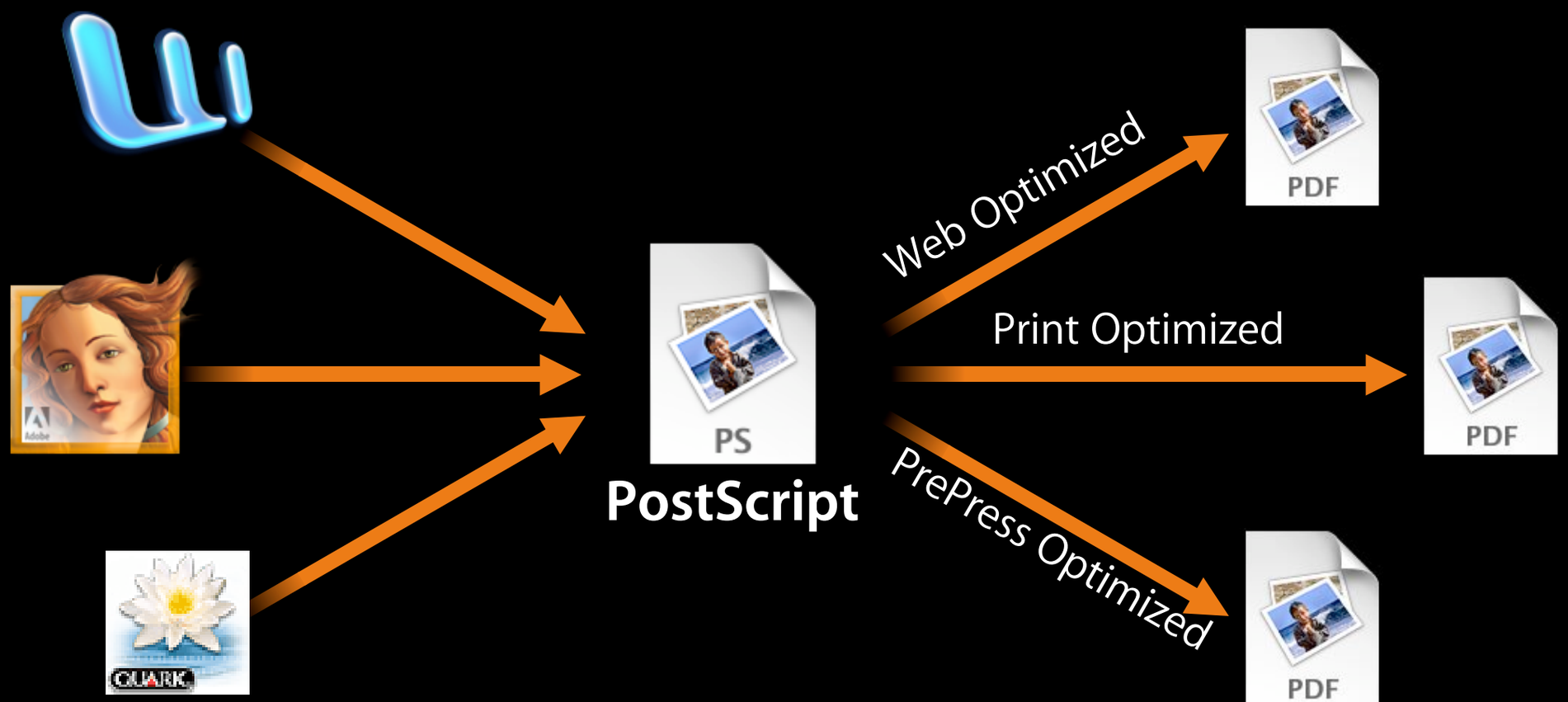
- Use PDF as Digital Paper:
  - Final form, paginated documents and vector artwork
- Use other formats, such as XML to encode models
  - Keynote file-format, others…
- Build a rich OS framework for Digital Paper
- Encourage Applications to differentiate by developing innovative models
- Long live MVC separation :-)

Graphic Arts Workflow



- Start with Application-dependent models
- Convert to application independent, device-independent marks on page
- Manipulate pages
- Convert to device-dependent representations

Traditional PS/PDF Workflow

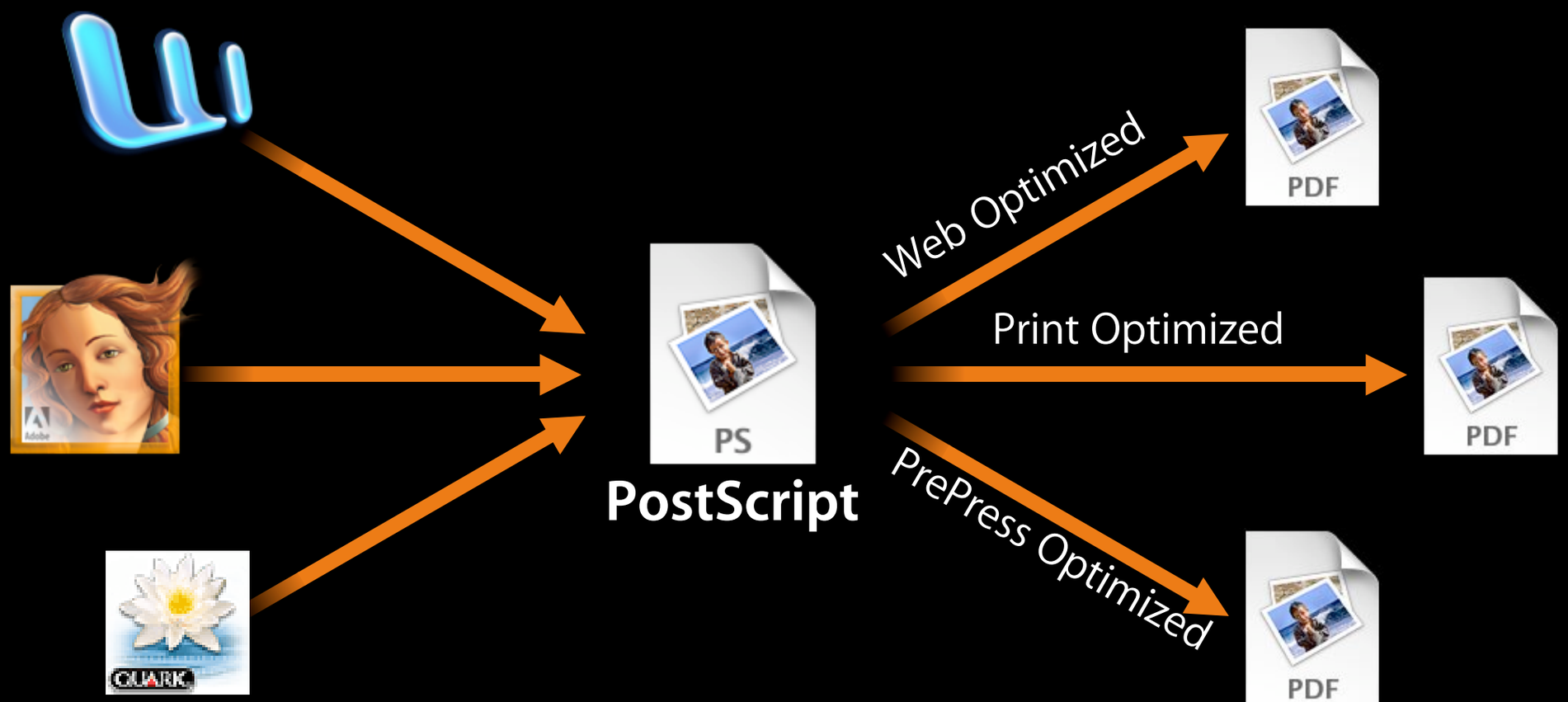


… But Wait

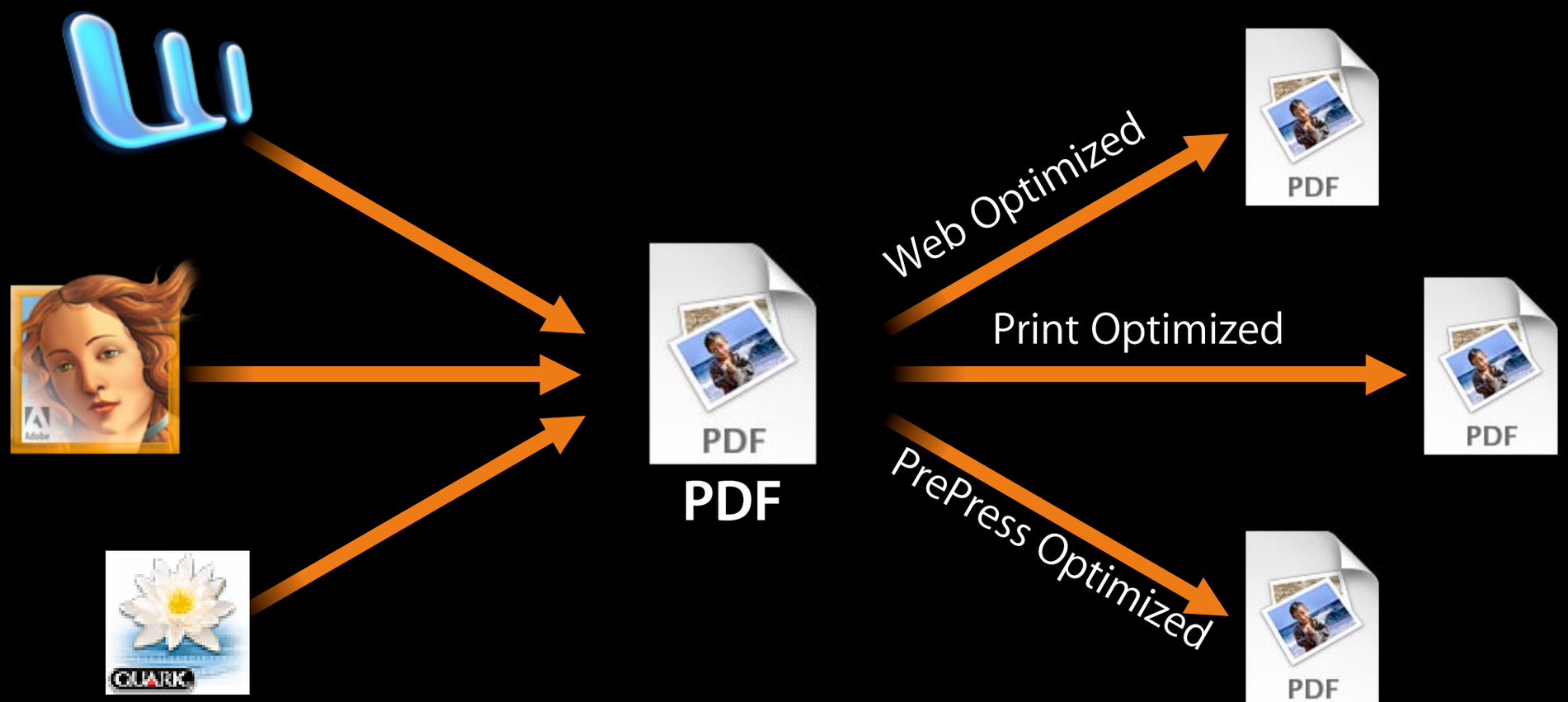
- Isn’t PDF device independent?
- What’s this “for Web” vs. “for Print” stuff?
- Suppose I create a “Web optimized” PDF and realize later that I need a “Print optimized” version?
- Do I keep the PS file around?
- Do we really want the PS file to be the application independent digital master?

NO!

Traditional PS/PDF Workflow

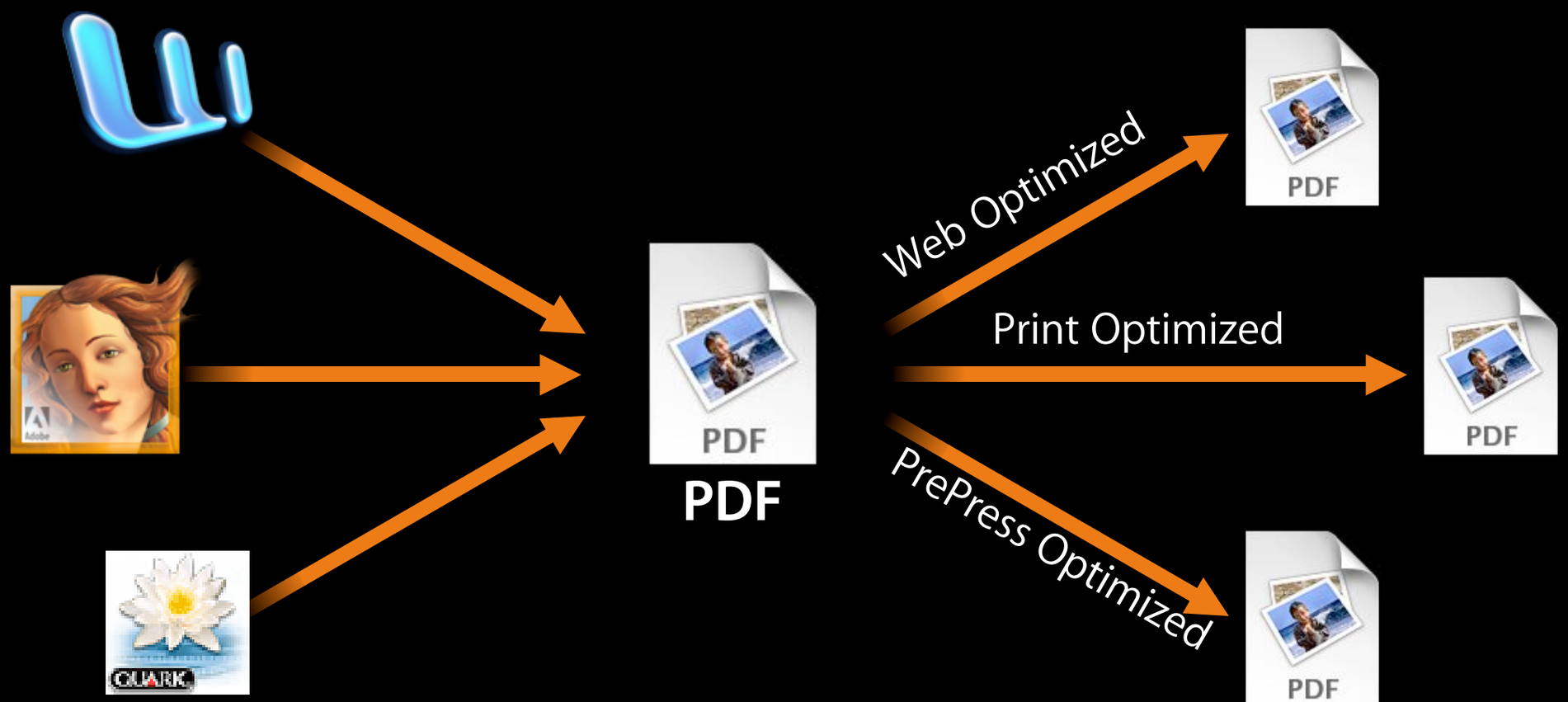


A Better Approach

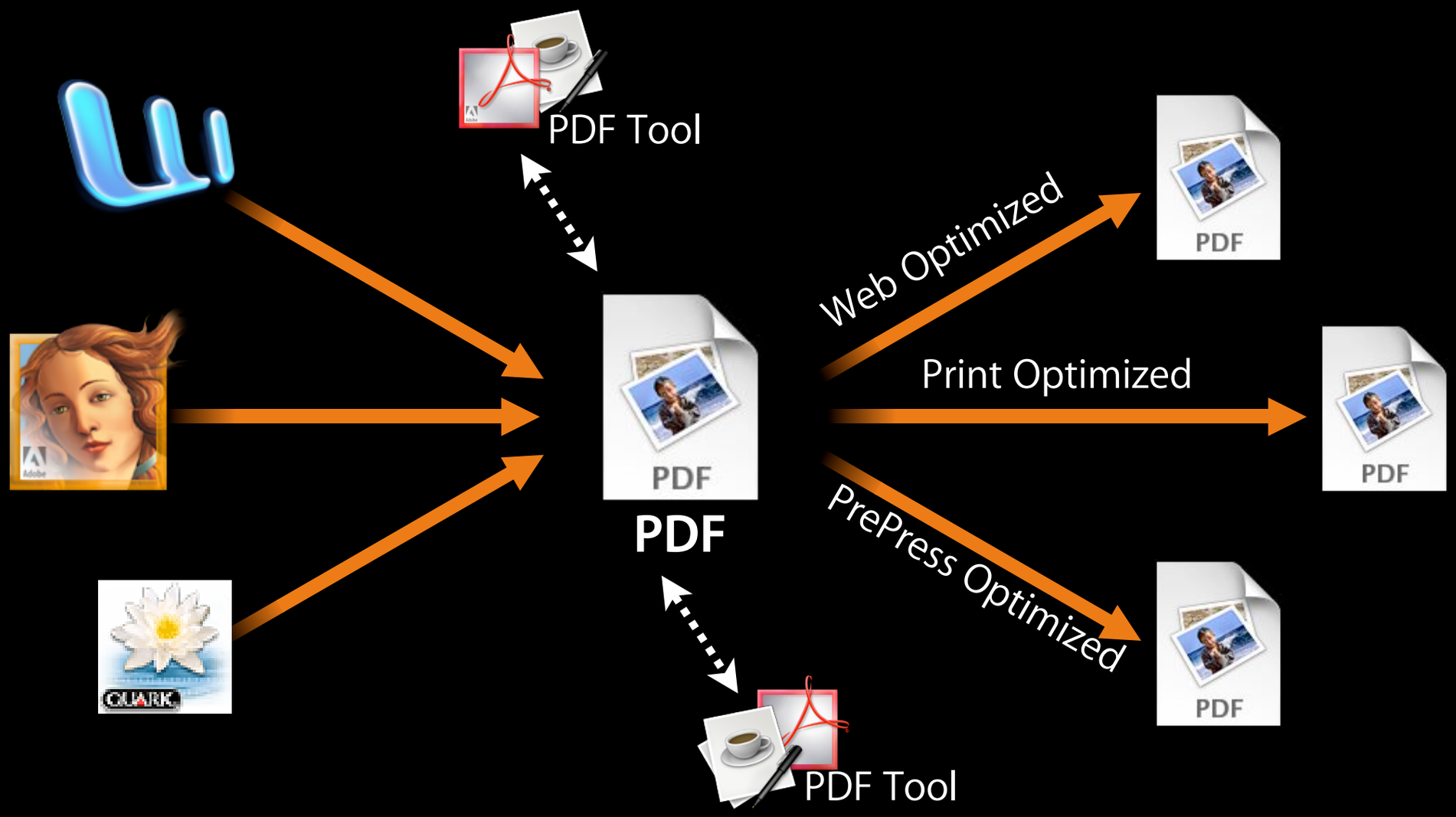


PDF Digital Paper

- Better suited “Digital Master” format than PostScript
- Viewable on 500M copies of Acrobat
- Lossless encoding of application intent
- “Digital Master” PDF does not pass judgement on application’s choice of:
  - Colorspace
  - Font technology
  - Image compression
  - Resolution


PDF
Web Optimized
PDF
Print Optimized
PDF
PDF
PrePress Optimized
PDF

Device-independent PDF Tools



PDF Processing on Mac OS X

- Version Tracker lists over 50 PDF applications on Mac OS X
- Panther will bring new opportunities

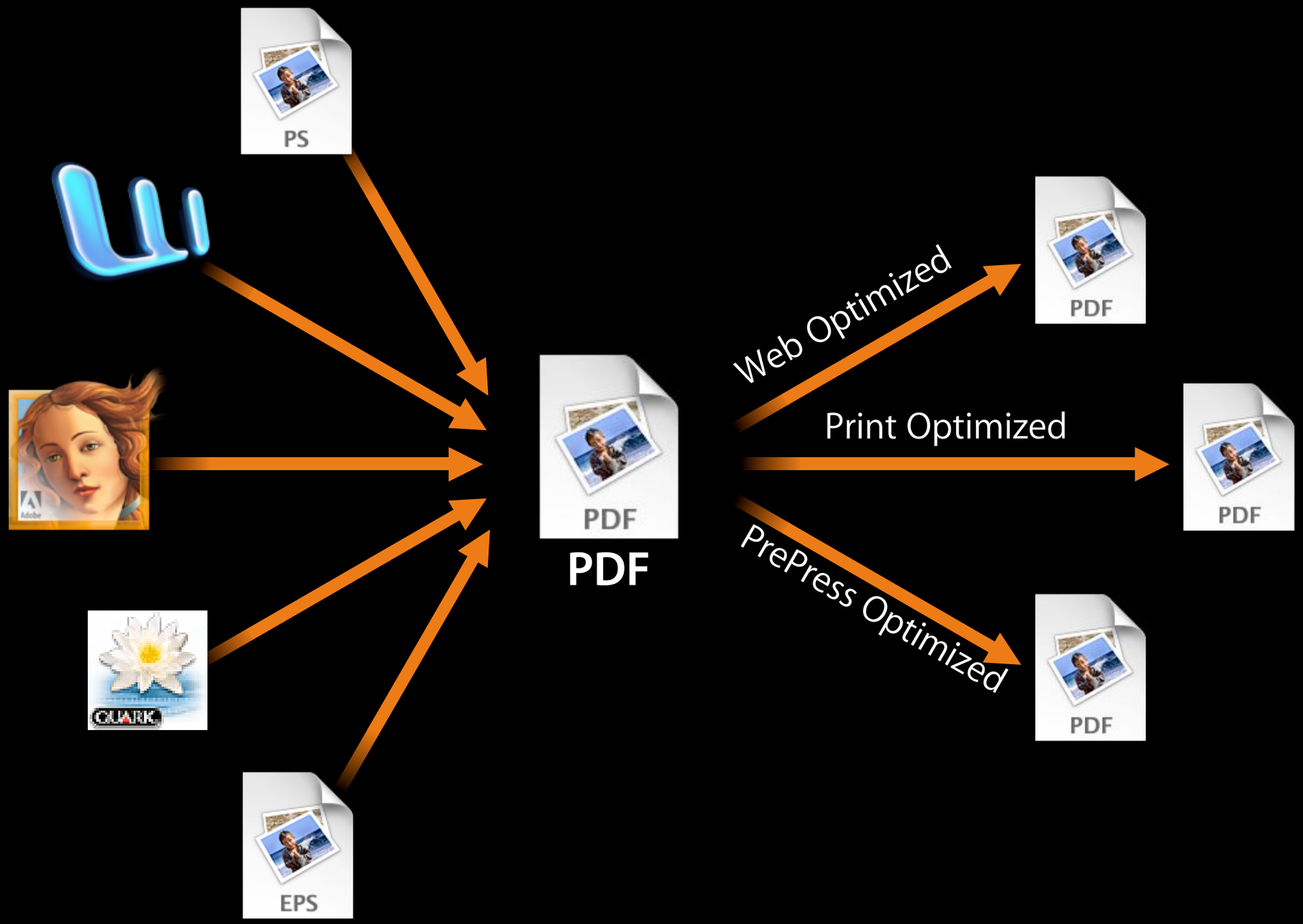
New PDF Workflow Toolset in Panther

- PostScript file handling
- User level scripting
- Quartz PDF filters
- Printing to a PDF workflow
- PDF Introspection APIs

PostScript to PDF Converter

- Gracefully deals with PS legacy files
- Provides “graphically lossless” transformation from PostScript to PDF
- Works for EPS as well
- Based on PostScript technology from Adobe
- Not a replacement for Distiller…
- Enables all printers to accept PostScript

PostScript Can Feed the Hub…



User-level Programmability

- Another lost aspect of PostScript legacy
- PostScript programming was useful for dynamically generating graphics
- PS program would “calculate” picture rather than “record” application drawing
- PDF can’t do this…
- … but reverse-polish notation was a real bear!

User-level Programmability

- Some folks found this aspect of PostScript useful…

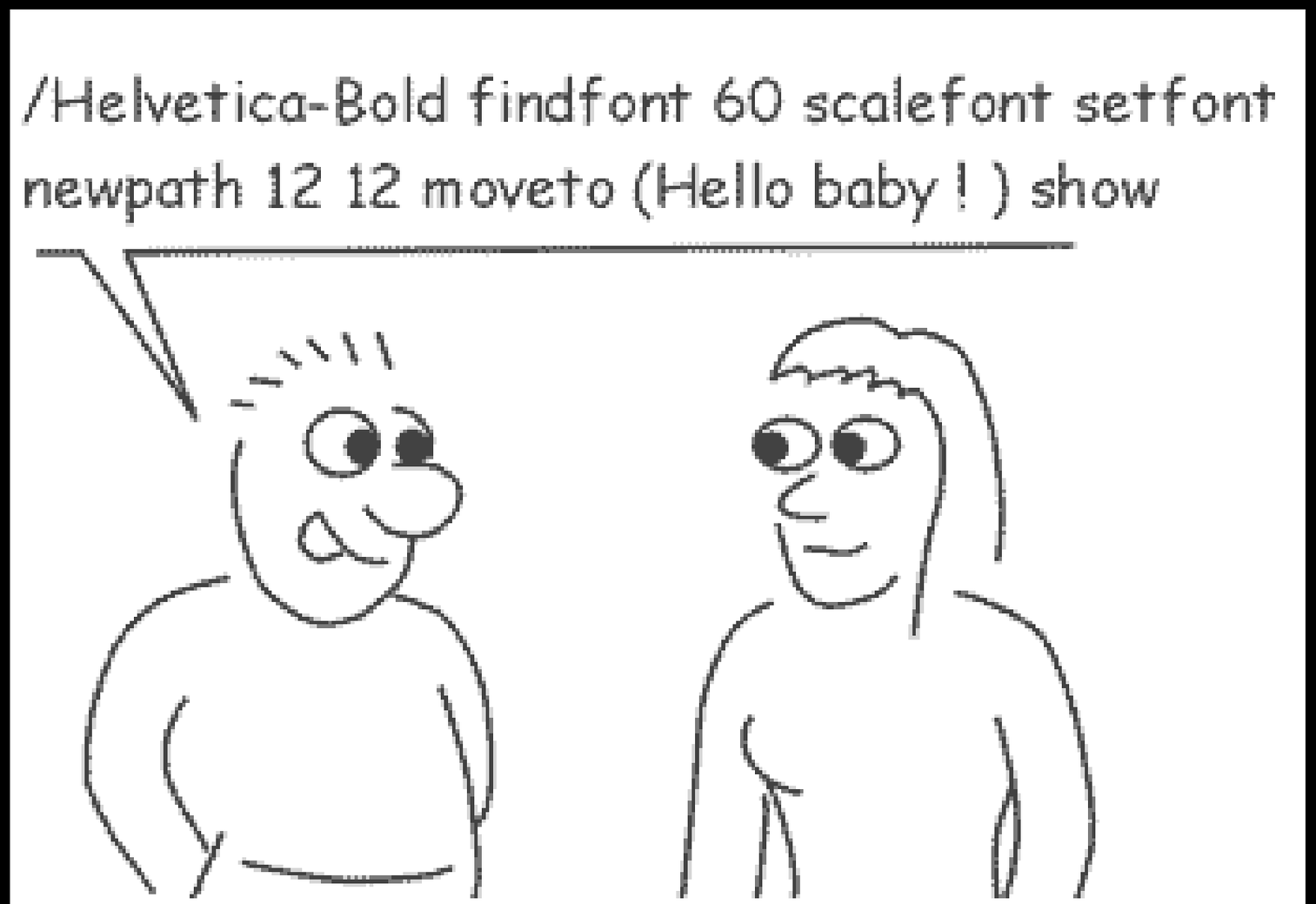


Image courtesy Laurens Leurs, http://www.prepressure.com

Quartz Scripting: Python Bindings for Quartz

- Full bindings for Quartz 2D API in Python
  - Allows PDF processing from scripts
- Convenience functions for:
  - Create CGImageRefs from files (via Quicktime Importer)
  - Save CGBitmapContext to files (via Quicktime Exporter)
  - Load HTML, RTF and Unicode Text (via Cocoa NSText)
  - Convert EPS and PS to PDF (via PS to PDF)
- Allows new generation of script writers to write simple tools

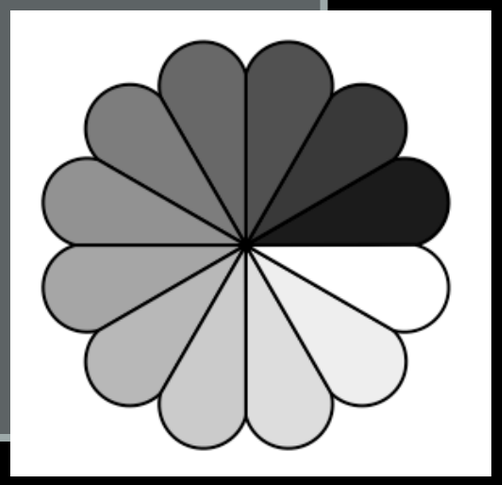
Quartz Scripting

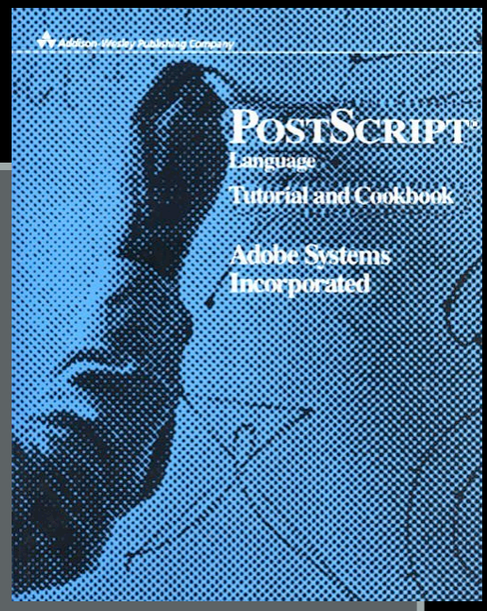
- Handy for small “one-off” processing scripts
- Rasterize PDF, HTML, RTF, PS documents into a bitmap and export via QuickTime
- Advanced imposition and page-to-paper mappings
- PDF concatenation
- Add a cover page, watermarks to a PDF
- Dynamic graphics generation on a server
- …unlike PS, can’t override marking operators
  - We consider this a feature :-)

Example

- PostScript Cookbook Wedge example

```
#!/usr/bin/python
from CoreGraphics import *
import math

#  For old times sake -- the PostScript cookbook wedge example
in python...

def inch(x):
	return 72*x

def rad(angle):
	return angle*math.pi/180.0

def wedge(c):
	c.beginPath()
	c.moveToPoint(0,0)
	c.translateCTM(1,0)
	c.rotateCTM(rad(16))
	c.translateCTM(0, math.sin(rad(15)))
	c.addArc(0,0, math.sin(rad(15)), rad(-90), rad(90), 0)
	c.closePath()

pageRect = CGRectMake (0, 0, 612, 792)   #  landscape
c = CGPDFContextCreateWithFilename ("wedge.pdf", pageRect)
c.beginPage(pageRect)
c.saveGState()
c.translateCTM(inch(4.25), inch(4.25))
c.scaleCTM(inch(1.75), inch(1.75))
c.setLineWidth(.02)
for i in range(1,13):
	c.setGrayFillColor(i/12.0,1.0)
	c.saveGState()
	wedge(c)
	c.fillPath()
	c.restoreGState()
	c.saveGState()
	c.setGrayStrokeColor(0,1)
	wedge(c)
	c.strokePath()
	c.restoreGState()
	c.rotateCTM(rad(30))
c.restoreGState()
c.endPage()
del(c)
```





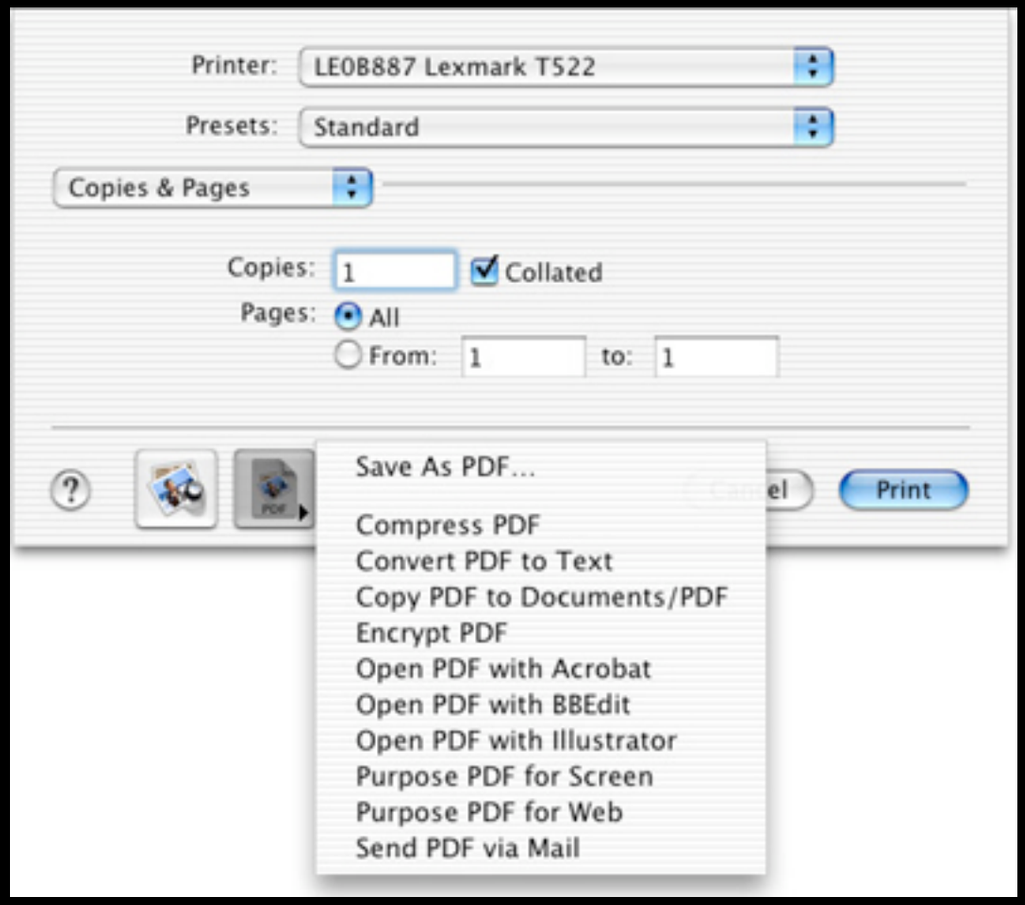

```
%!PS-Adobe-3.0
%%BoundingBox: 172 173 259 260

/inch {72 mul} def

/wedge
{ newpath
  0 0 moveto
  1 0 translate
  15 rotate
  0 15 sin translate
  0 0 15 sin -90 90 arc
  closepath
} def

gsave
  3 inch 3 inch translate
  0.5 inch 0.5 inch scale
  0.02 setlinewidth
  1 1 12
  { 12 div setgray
    gsave
      wedge
      gsave
       fill
      grestore
      0 setgray
      stroke
    grestore
    30 rotate
  } for

grestore

showpage
```

Quartz PDF Filters

- Allows processing of PDF transformations based on “recipes”
- “Recipes” are created in ColorSync Utility
- Can be used for color conversion or effects
- Image resampling and compression is also an option

Print-to-PDF Workflow

- Workflow items are placed here:
  - ~/Library/PDF Services, /Library/PDF Services
- Supported Workflow items
  - Folder
  - Folder alias
  - Application or application alias
  - CLI tool
  - AppleScript
  - Quartz Filter (new on Panther)



PDF Introspection APIs

- Complete access to PDF document structure
- Dictionaries, Strings, Arrays, Streams
- Does not model internals of GraphicsStreams
- Useful for extracting links, annotations, metadata, etc.

Demo

Preview and PDF Workflow

Quartz and PDF Summary

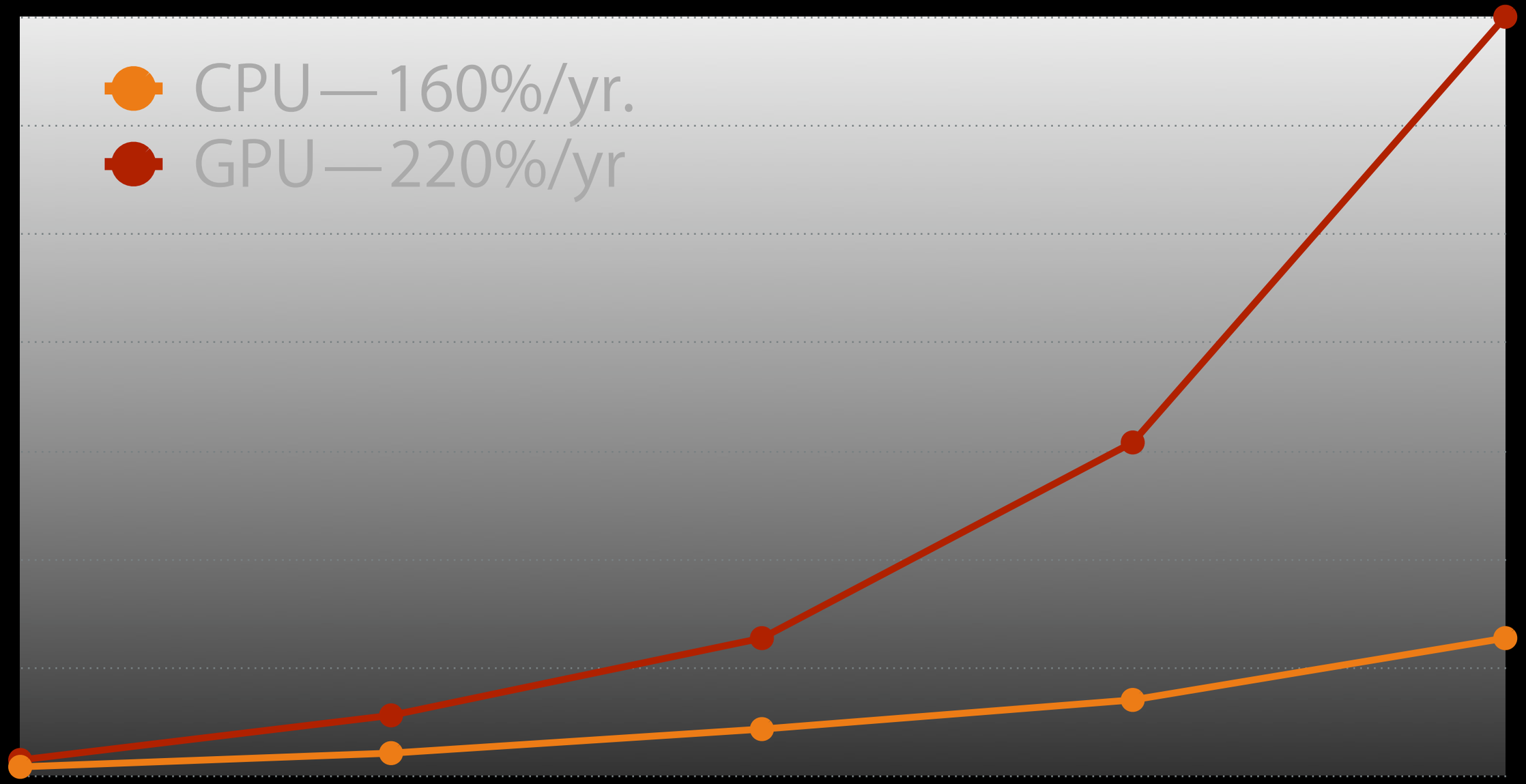
- PDF provides Mac OS X with a universal representation of “Digital Paper”
- Apple’s strategy is to continue to build upon the “view” level abstraction of PDF
- Panther adds significantly to the PDF toolset
- If you’re still thinking PixMaps and GWorlds, come join the Quartz 2D party!

The GPU Computing Revolution

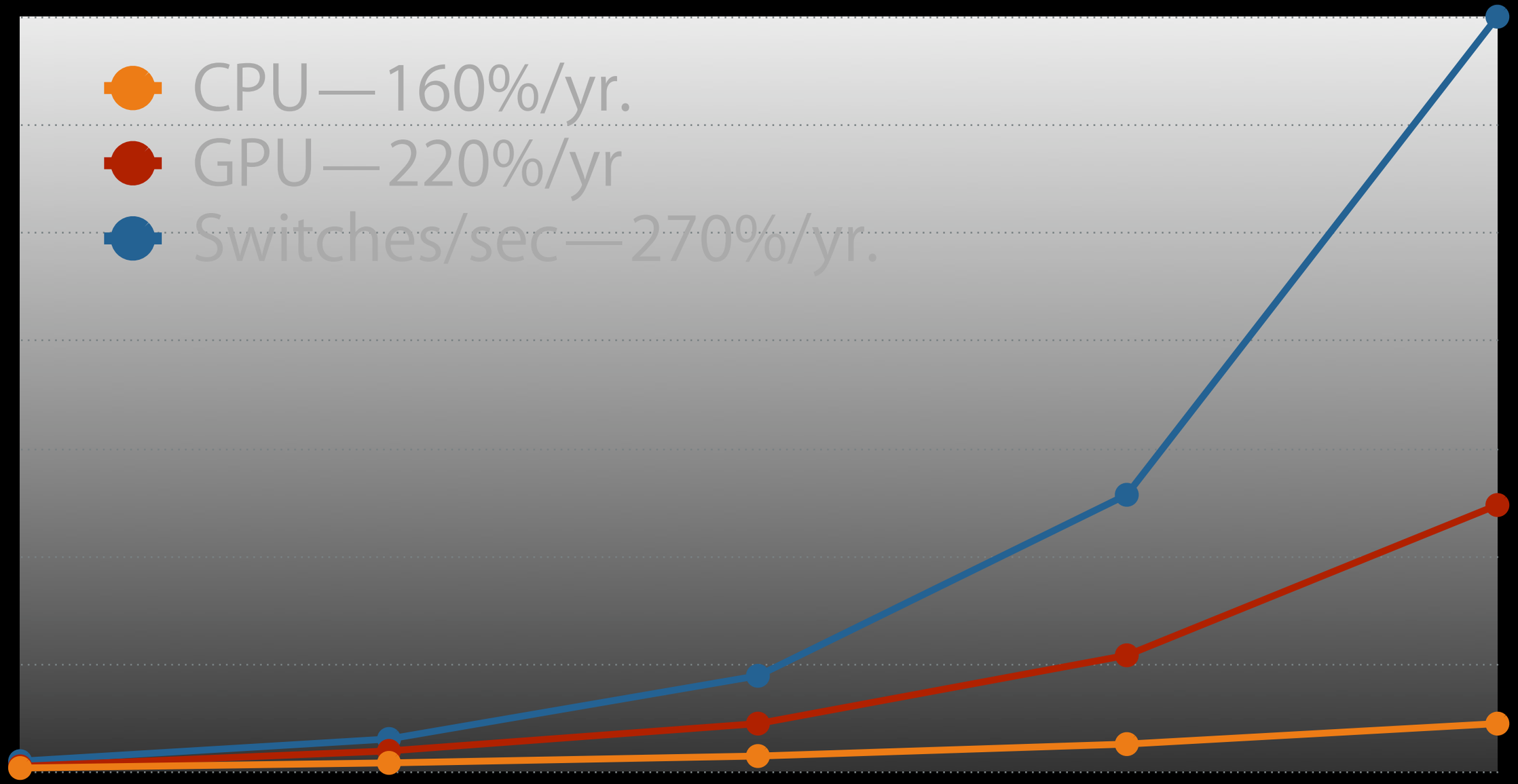
First Some Numbers…

- GPUs can composite images faster than CPUs
- GPU:
  - 8 pixels per clock @ 400 MHz
  - 21.8 GB/s memory bandwidth
- CPU:
  - ~1 pixel per clock @ 2 GHz
  - 6.4 GB/s memory bandwidth
- Frees CPU for application work

CPUs vs. GPU Performance



“Capability” Growth Even Higher…



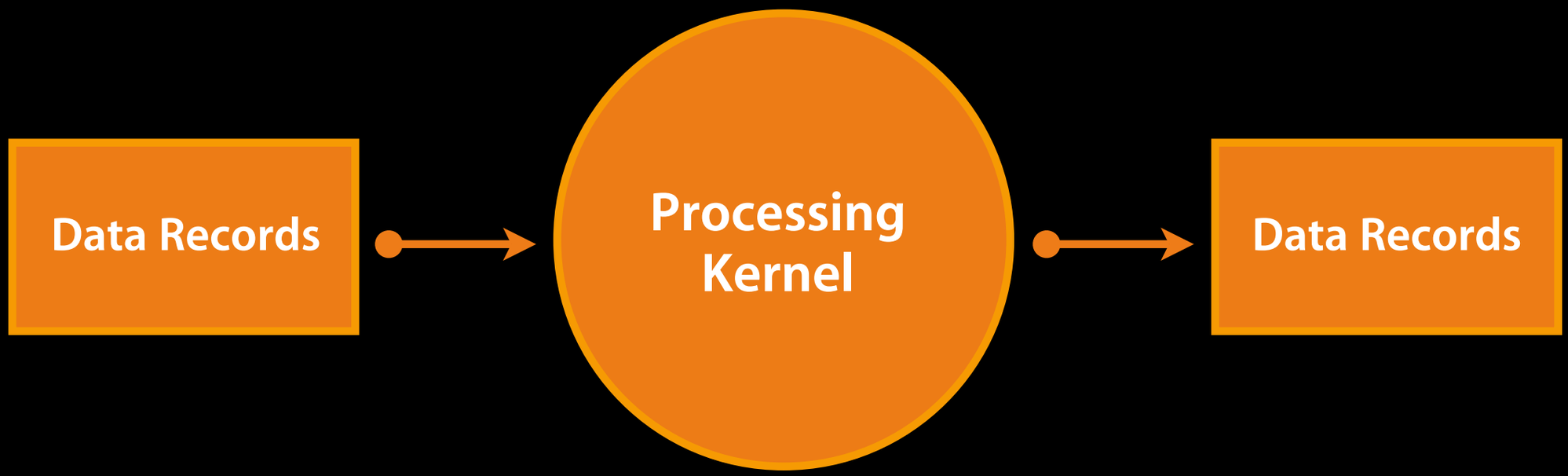
Why Is This So?

- What do GPU designers know that CPU designers don't?
  - Answer: Nothing!
- GPUs are not general purpose processors
  - Computation is "embarrassingly parallel"
- Entertainment applications (i.e., games) have provided a sustainable economic model for mass market parallel computing

Computational Intensity

- Arithmetic intensity / switch
  - GPU: double the gates => double the performance
  - CPU: double the gates => much less
- General-purpose CPUs dedicate a lot of resources to non-computational logic:
  - Instruction and data caches
  - Branch prediction
  - Instruction reordering
  - Data hazard avoidance

Stream Processing Model



Performance Benefits of Stream Processing

- A GPU can be thought of as 1000s of stream processors
- Get input from another stream processor
- Send output to next stream processor
- High-speed dedicated producer/consumer links
- Stream processors stays busy all the time
- Most stream units are dedicated logic…
- … programmability is emerging

The Graphics Pipeline

Primitive Submission

Vertex Processing

Triangle Rasterization

Fragment Processing

Blending, z-test, etc.

The Graphics Pipeline

Primitive Submission

- Convert Primitives into vertex stream
  - "Vertex" is a state bundle including:
    - Position vector
    - Normal vector
    - Color vector
    - Texture coordinate vectors
    - Actually up to 16 vector attributes

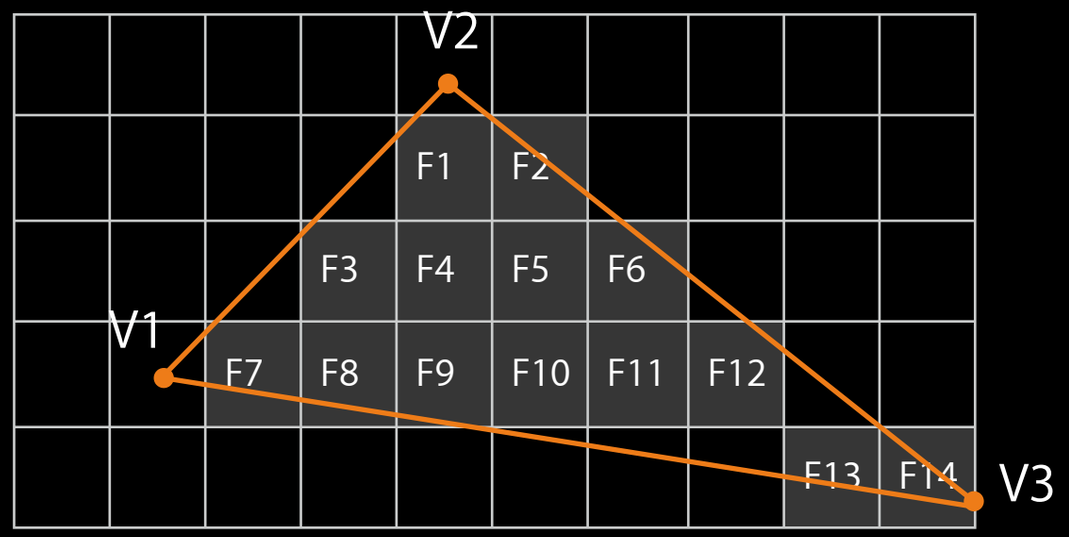
The Graphics Pipeline

Vertex Processing

- Vertices are processed
  - Spatial vectors transformed
  - Lighting calculated
  - Texture coordinates get generated and transformed

The Graphics Pipeline

Triangle Rasterization



Vertex Attributes interpolated over interior of triangle

The Graphics Pipeline

Fragment Processing

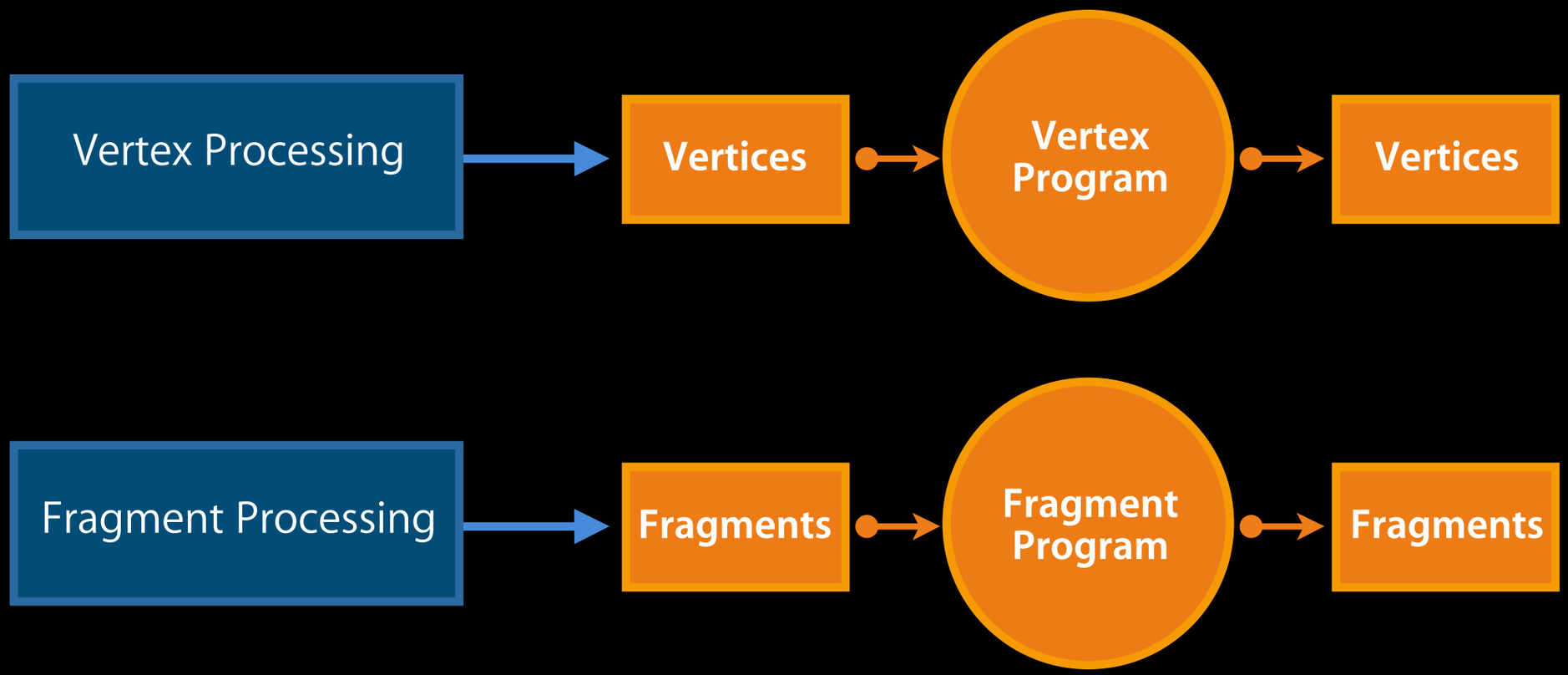
- Fragments are processed
  - Texture mapping applied
  - Final color and z calculated

The Graphics Pipeline

Blending, z-test, etc.

- Fragments are rendered
  - z-testing against framebuffer
  - Alpha-blending with framebuffer
  - Fog, etc.

The Programmable Graphics Pipeline



Pixel Programming on the GPU

- Data model similar to AltiVec
  - 128 bit 4-vectors of floats (e.g., RGBA, XYZW)
- Small program executed to calculate each output pixel
- Access to iterated vertex attributes
- Memory reads (texture lookups)
- Can’t read destination pixel
- Output left in special “result” register
- Powerful ALU and swizzling instructions

Pseudo Code for Pixel Program

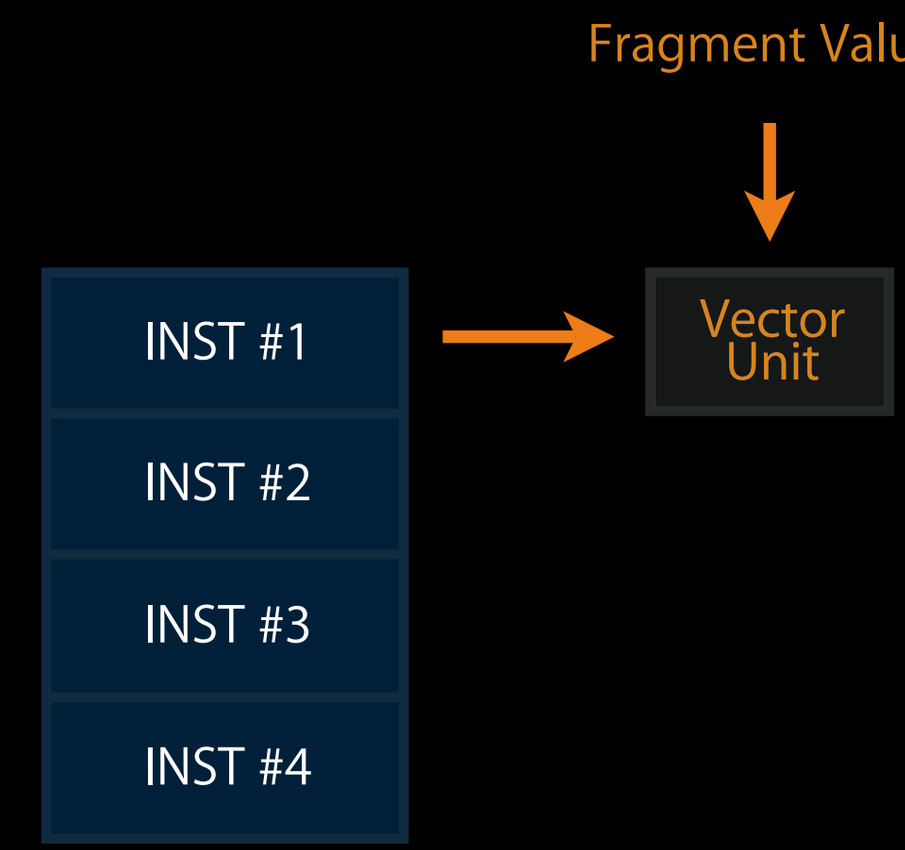
- All have the same basic function signature

```
global const vector globals[];  // vector constants
global vector textures[];       // pointers to textures

// return result based on interpolated values
vector  calculatePixel(const struct iteratedValues)
{
   vector result;
   result = iteratedValues.color;
   return result;
}
```

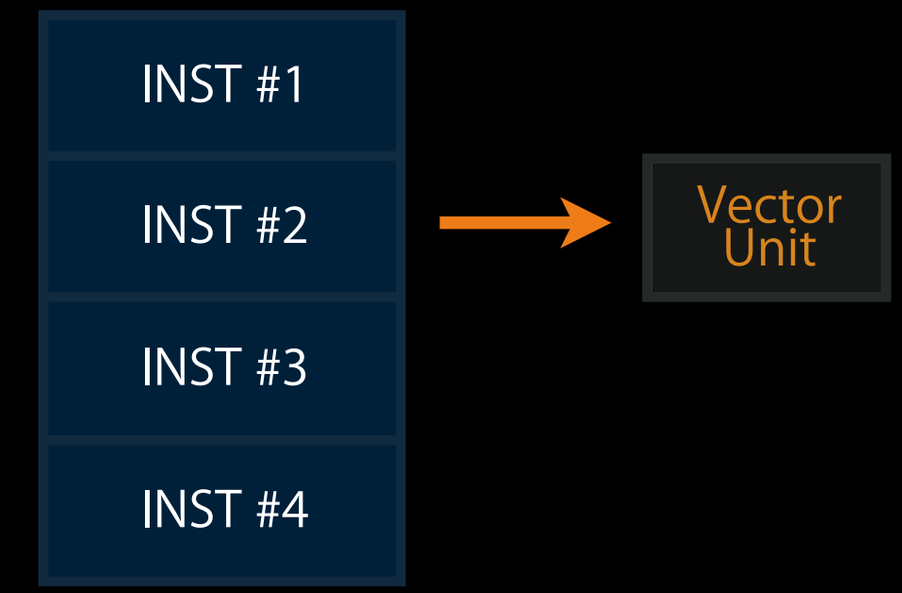
Disclaimer!

- Hardware design is hard
- The following examples are illustrative of the kinds of speed-ups possible with the constrained computational model

Sequential Processing



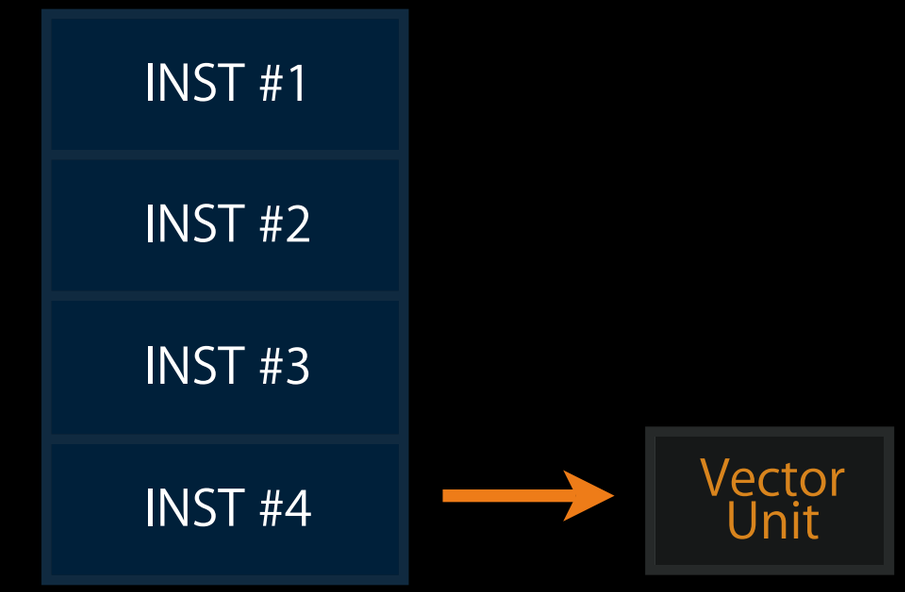
Sequential Processing



Sequential Processing

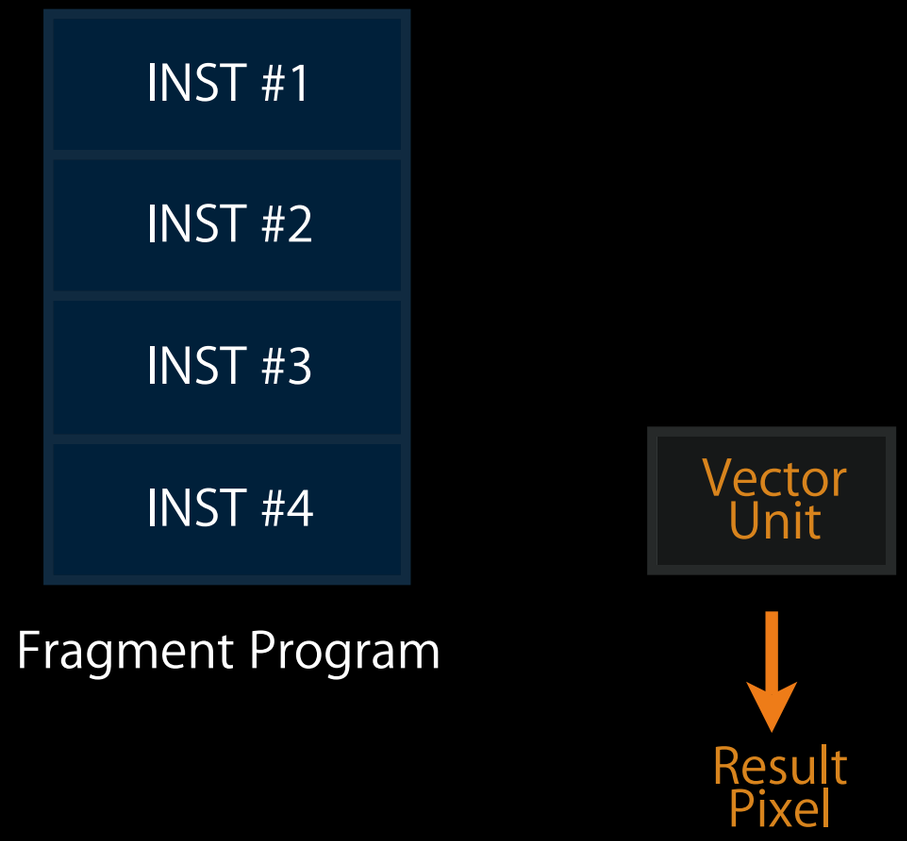
Fragment Program

Sequential Processing



Fragment Program

Sequential Processing



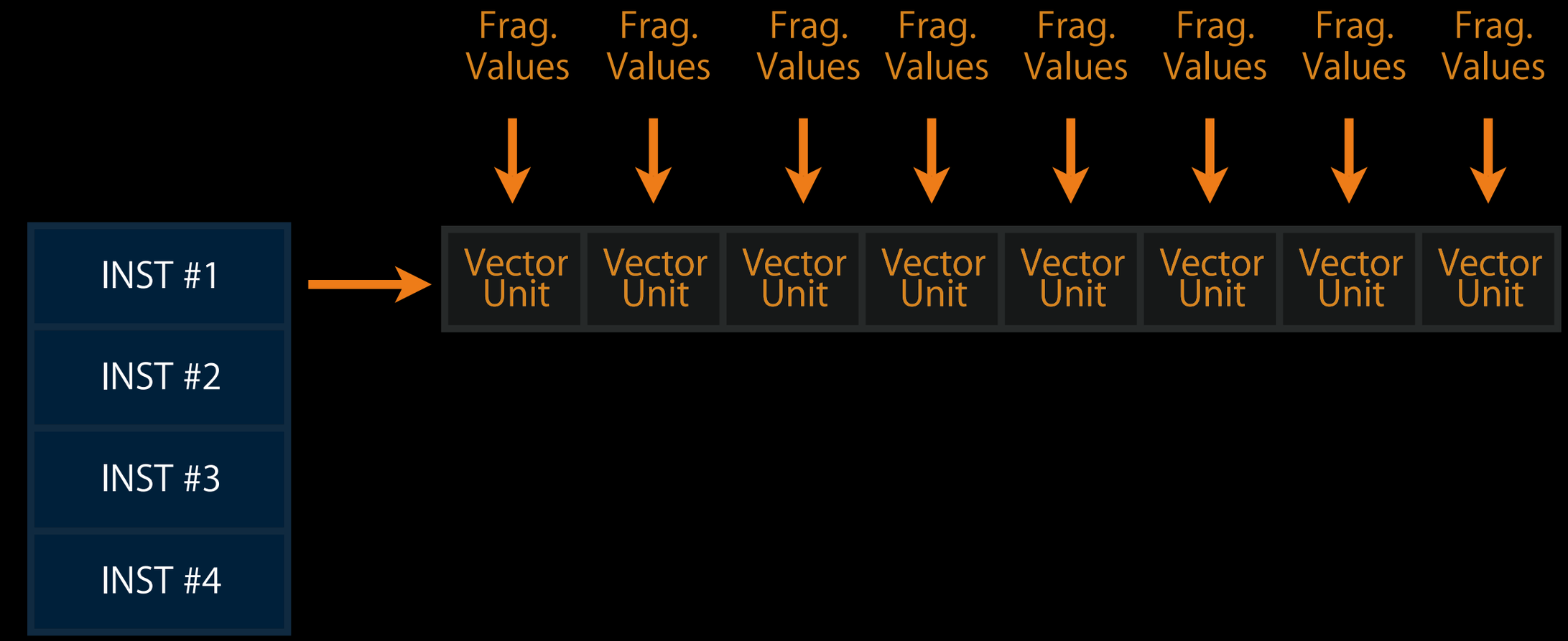
1 vector operation per clock
1 result every 4 clocks

Data Level Parallelism

- Pixel calculations are independent
- ResultPixel(i,j) cannot depend on ResultPixel(k,l)
- Pixel calculations can execute in parallel
- Call this parallelism in “space”
- Replicate the fragment execution unit and perform the same instruction on multiple pixels at once

Parallel Processing

Parallelism in “space” allows wide execution



Fragment Program

Parallel Processing

Parallelism in “space” allows wide execution

Parallel Processing

Parallelism in “space” allows wide execution

Fragment Program

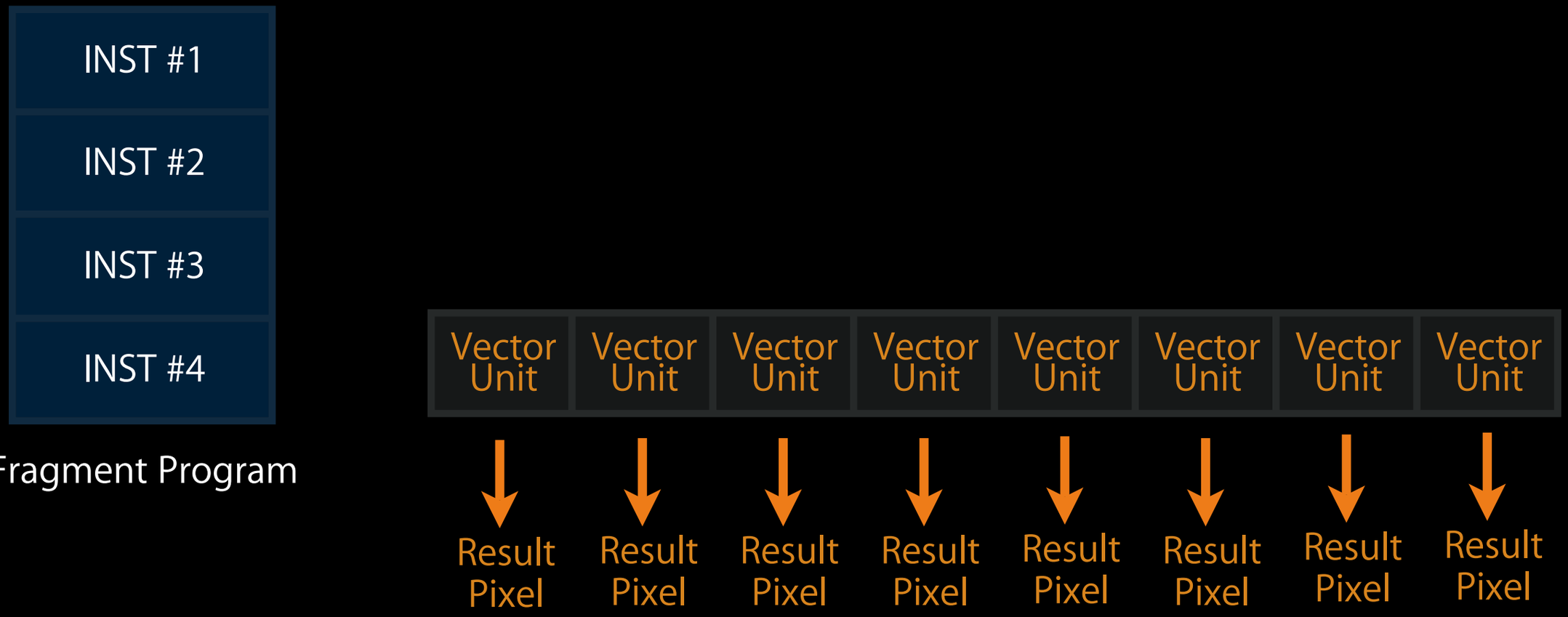
Parallel Processing

Parallelism in “space” allows wide execution

Fragment Program

Parallel Processing

Parallelism in “space” allows wide execution



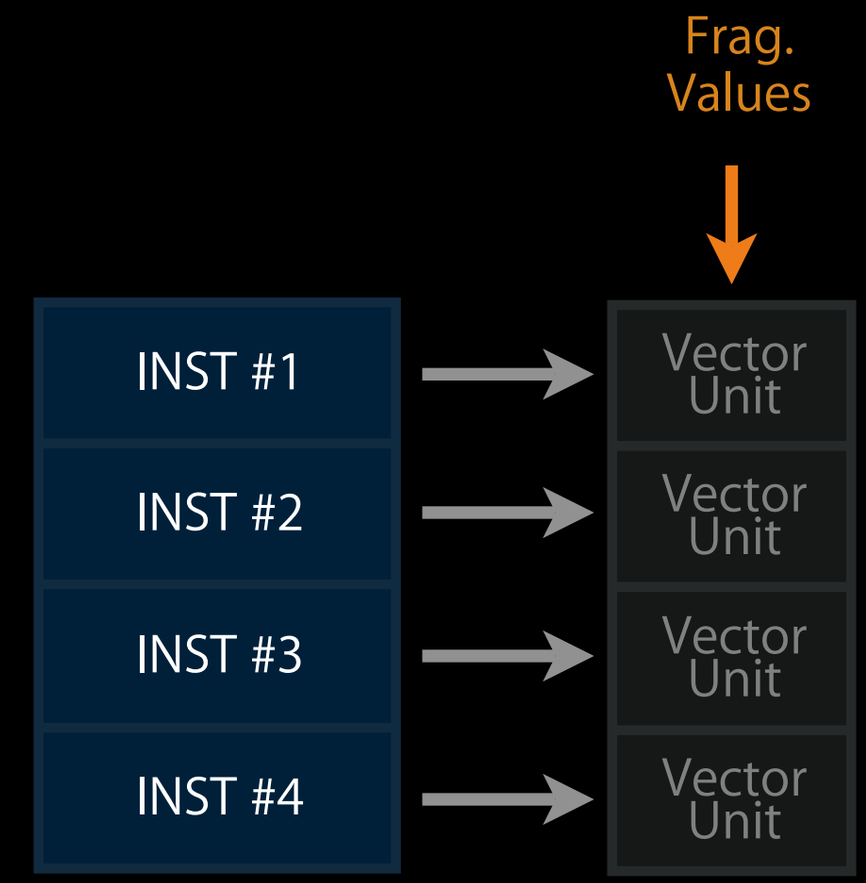
8 vector operations per clock
8 results every 4 clocks

Instruction Level Parallelism

- No writeable memory
- Machine state can only differ by one register from one instruction to the next
- Register renaming can hide instruction dependencies
- Code sequences are small and branchless
  - Static analysis at compilation time possible
- Physical registers > architecture registers

Parallel Processing

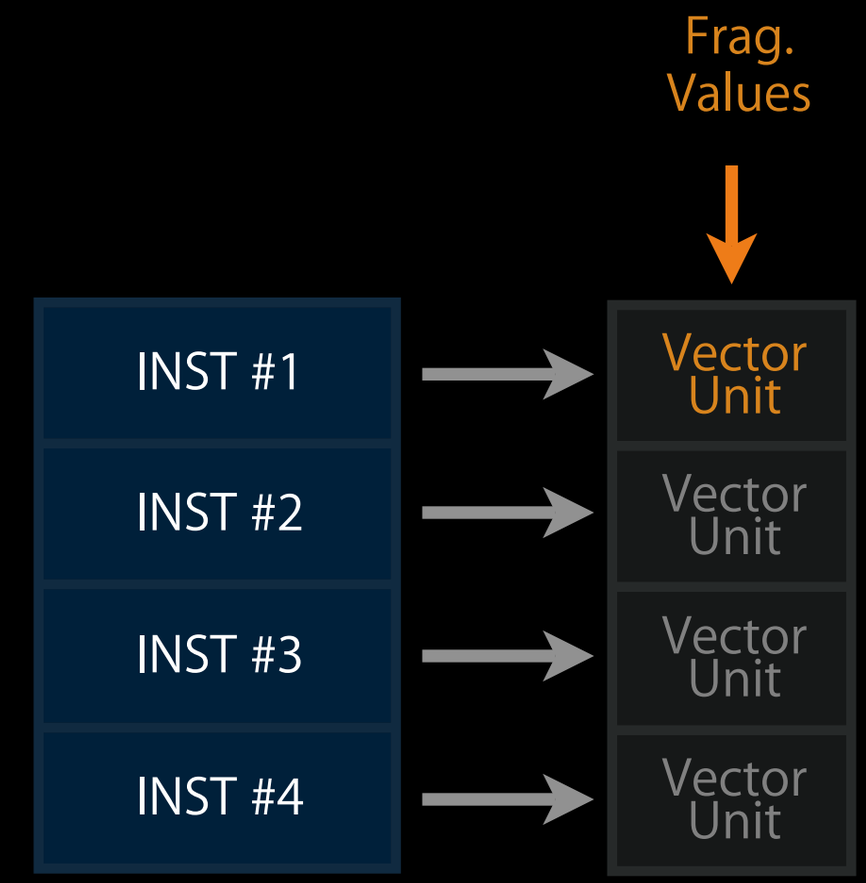
Parallelism in “time” allows pipelined execution



Fragment Program

Parallel Processing

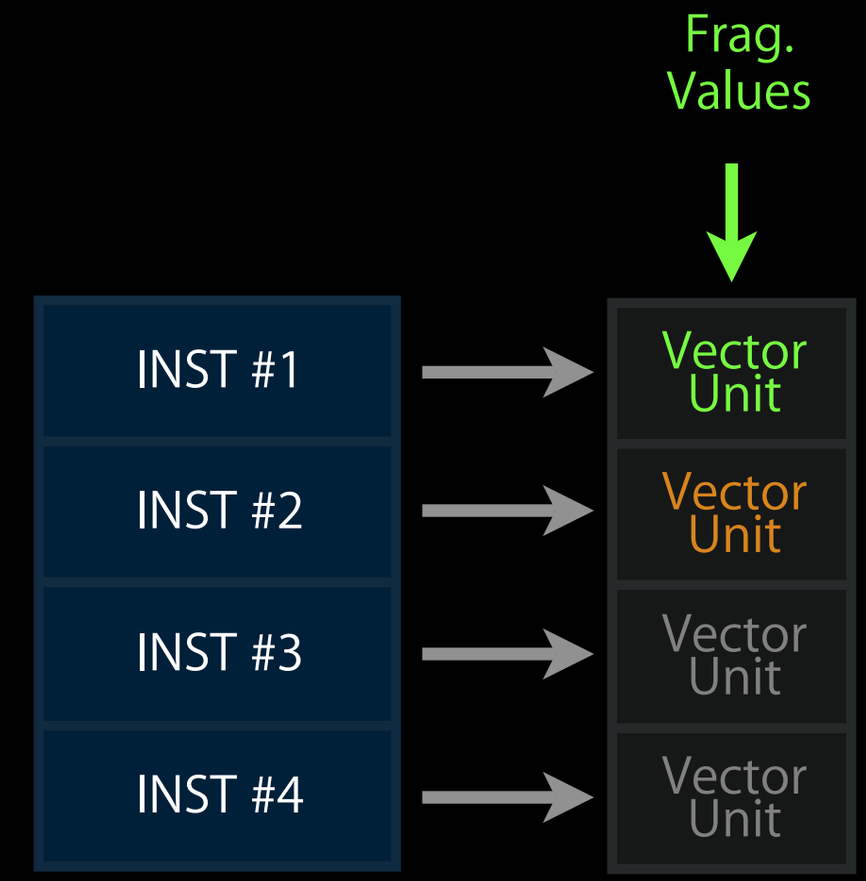
Parallelism in “time” allows pipelined execution



Fragment Program

Parallel Processing

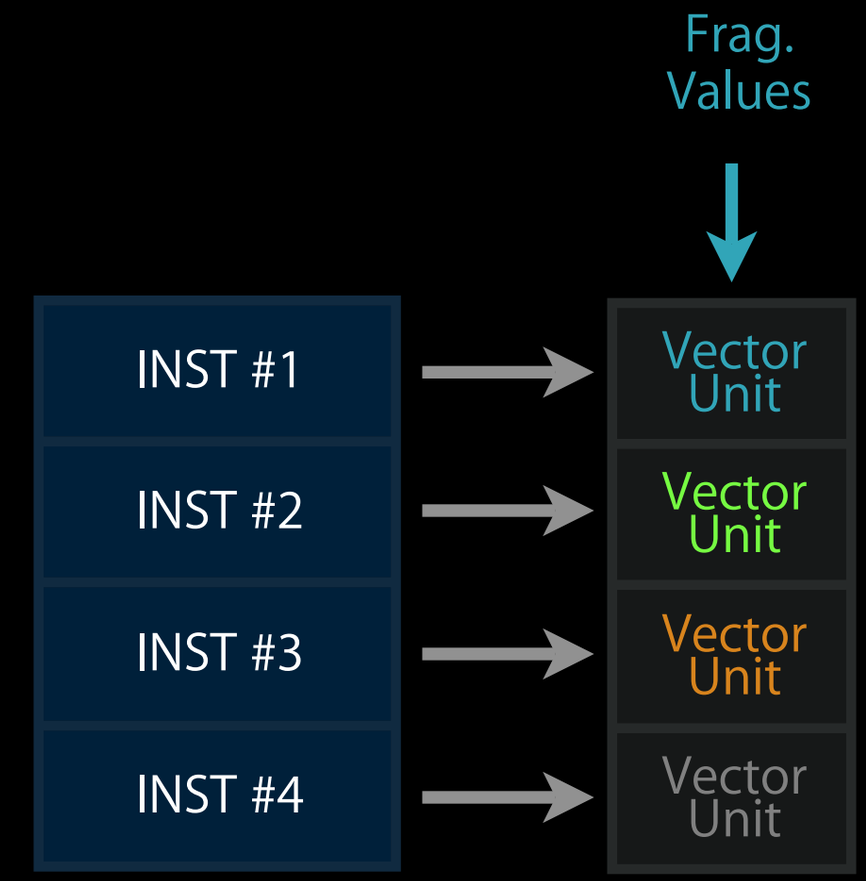
Parallelism in “time” allows pipelined execution



Fragment Program

Parallel Processing

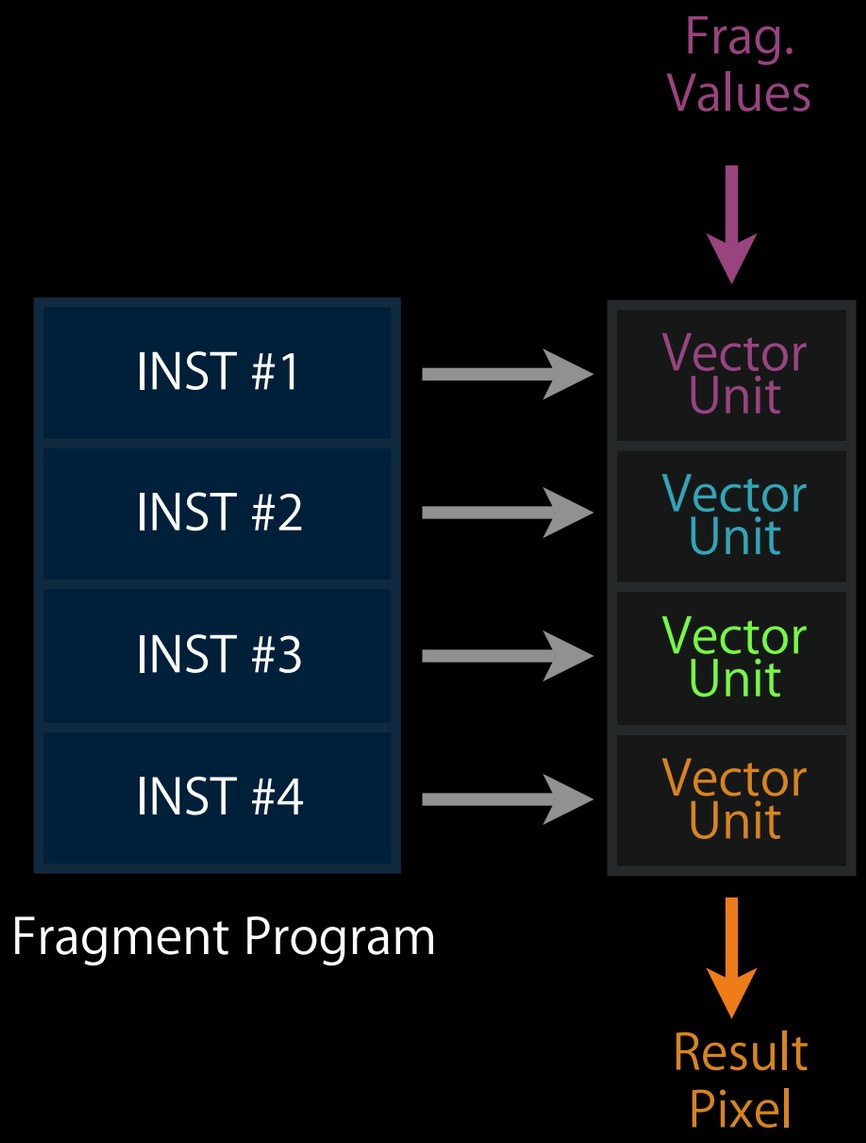
Parallelism in “time” allows pipelined execution



Fragment Program

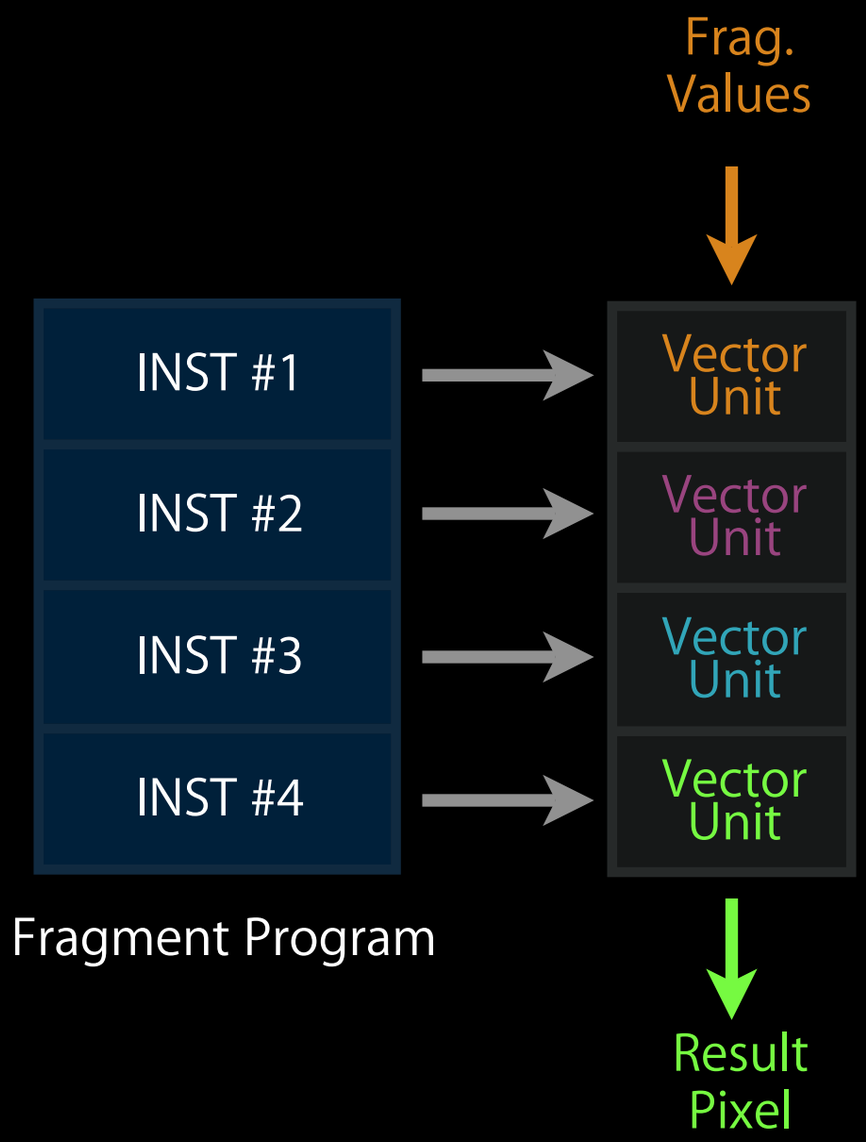
Parallel Processing

Parallelism in “time” allows pipelined execution



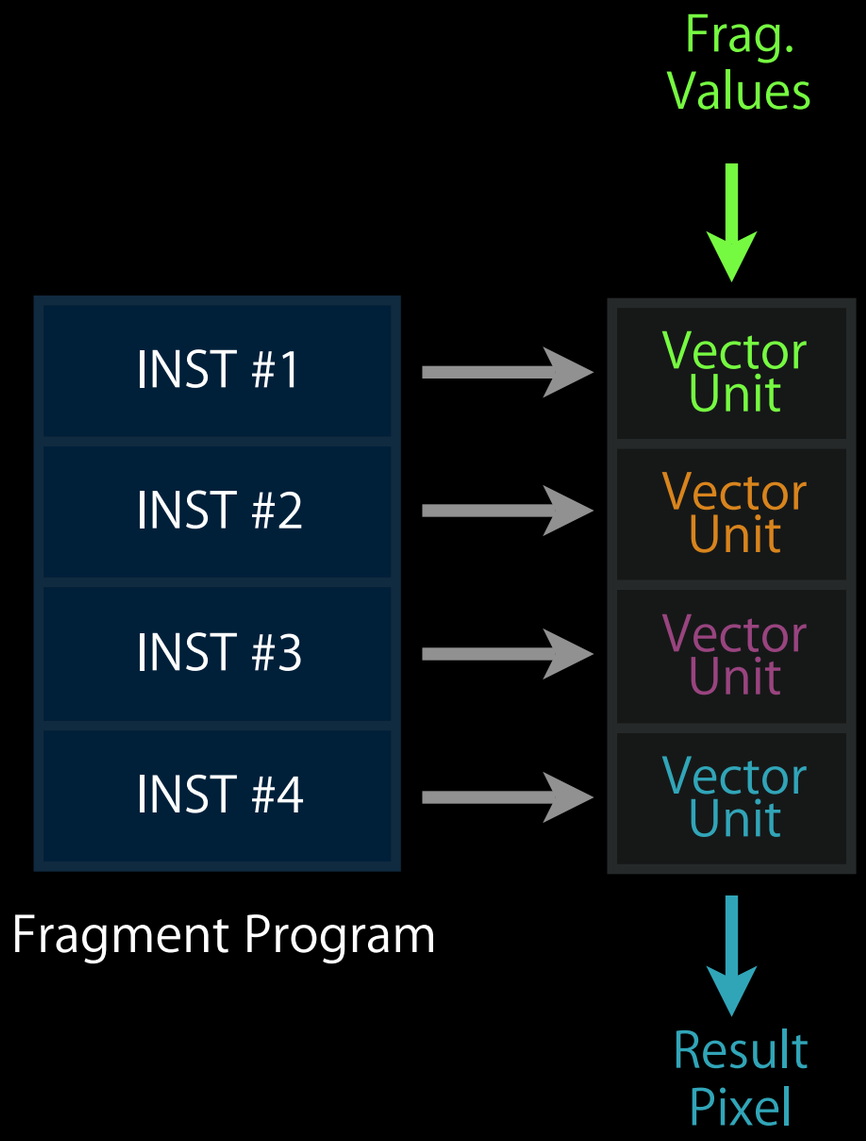
Parallel Processing

Parallelism in “time” allows pipelined execution



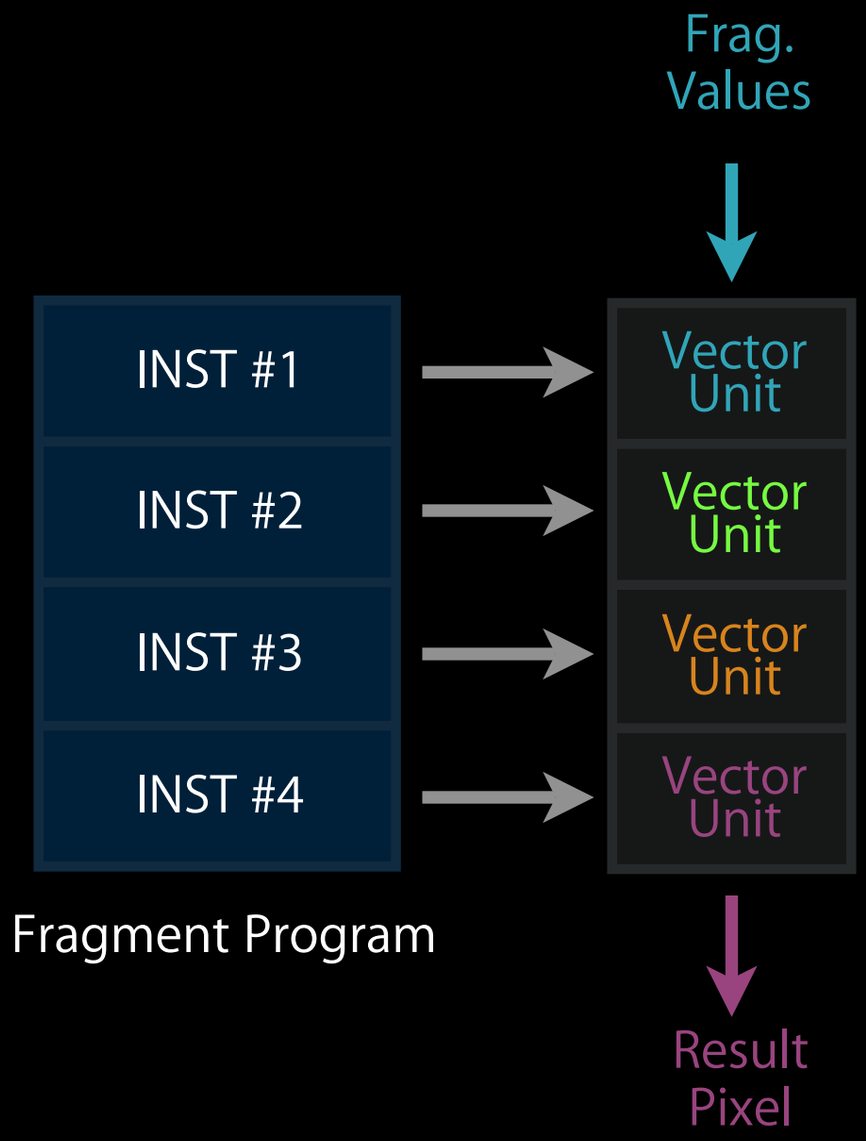
Parallel Processing

Parallelism in “time” allows pipelined execution



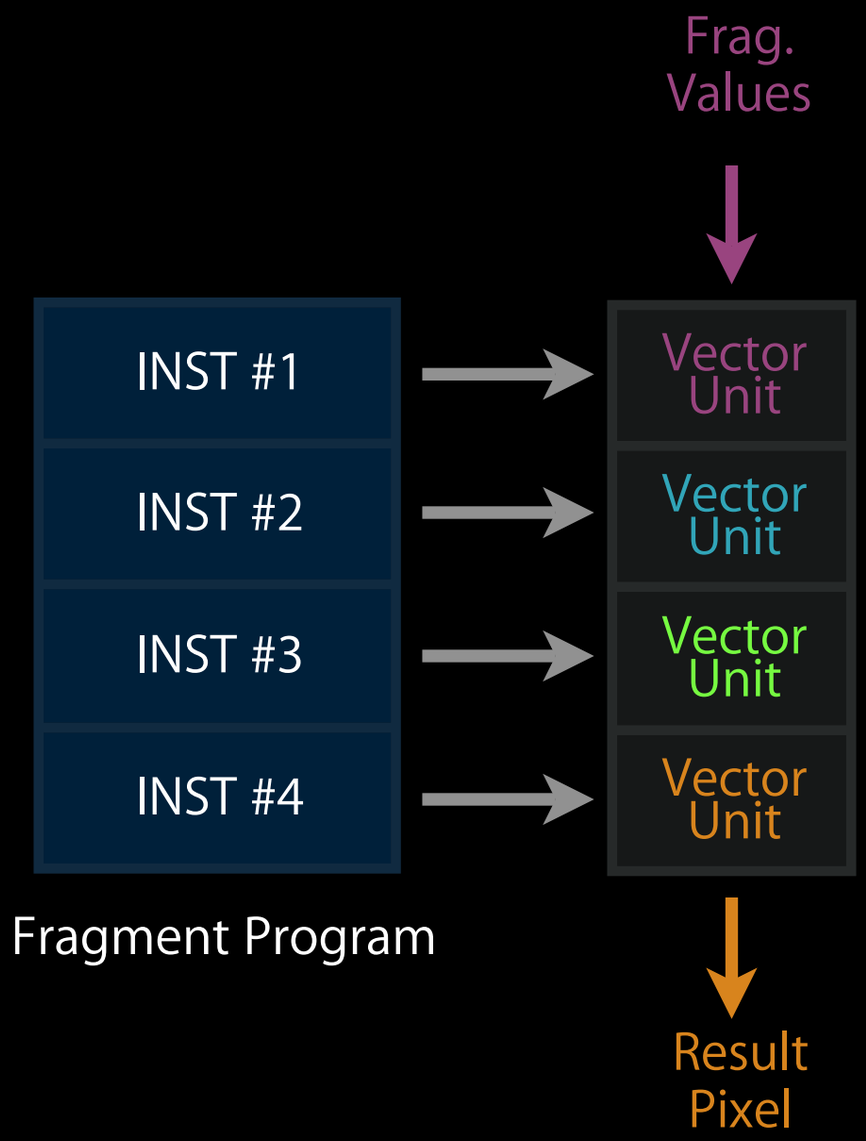
Parallel Processing

Parallelism in “time” allows pipelined execution



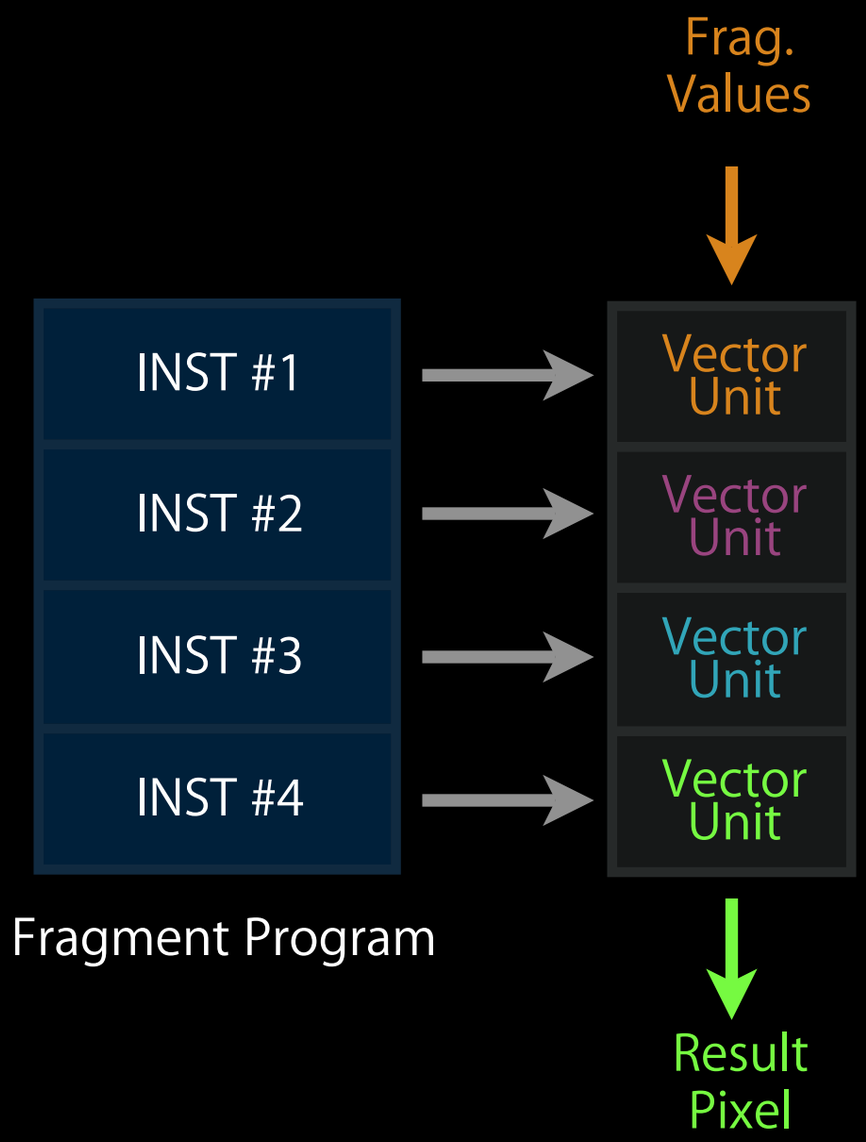
Parallel Processing

Parallelism in “time” allows pipelined execution



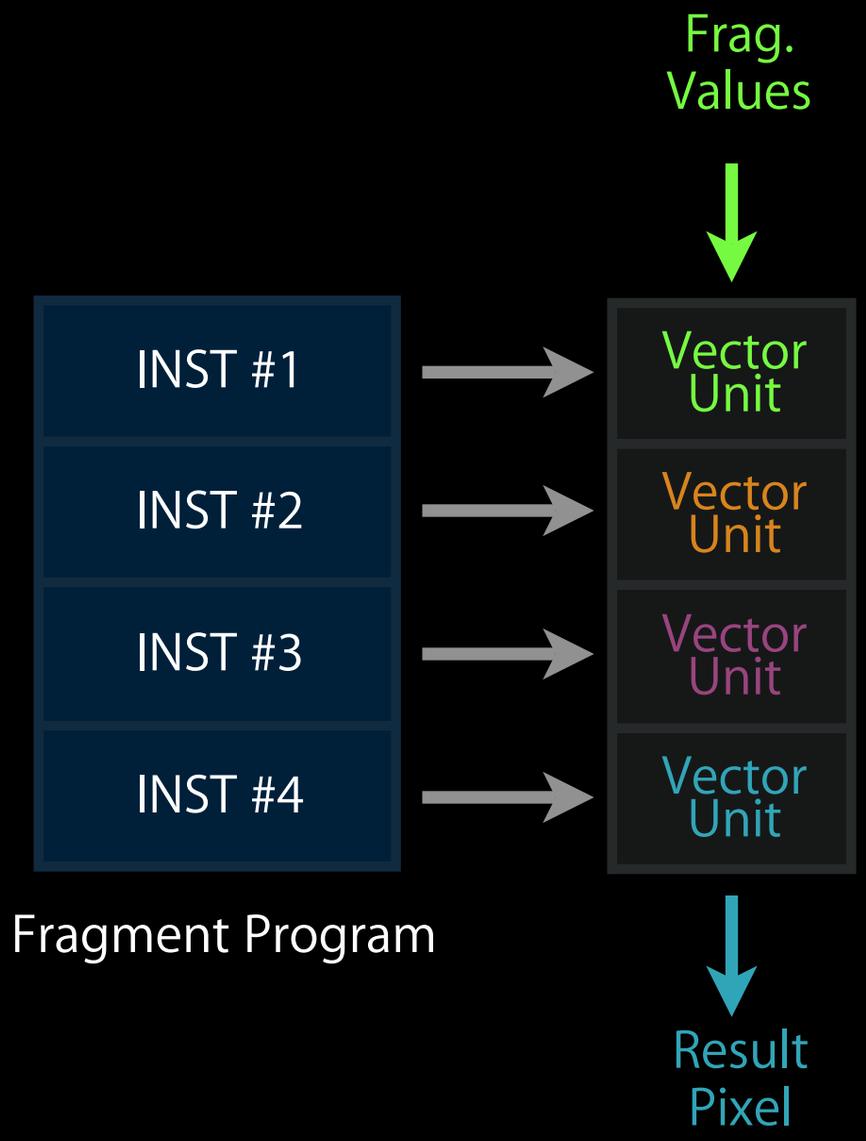
Parallel Processing

Parallelism in “time” allows pipelined execution



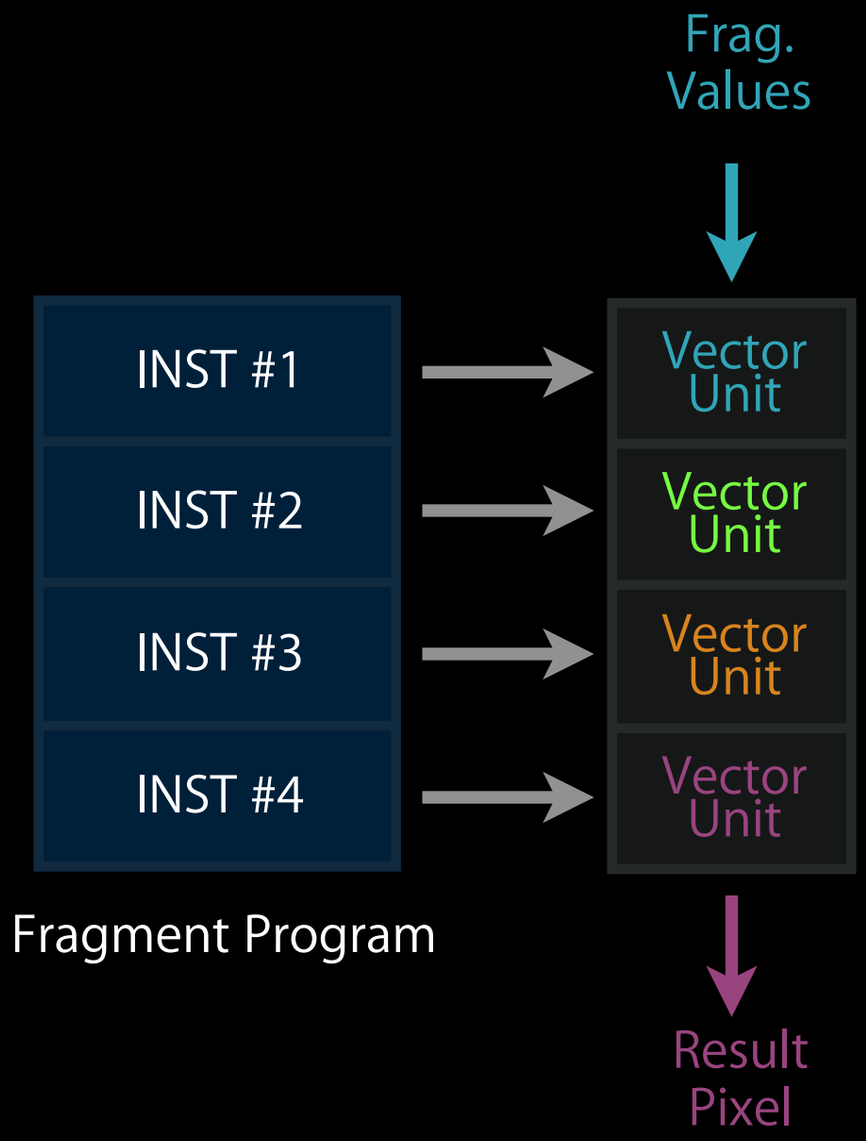
Parallel Processing

Parallelism in “time” allows pipelined execution



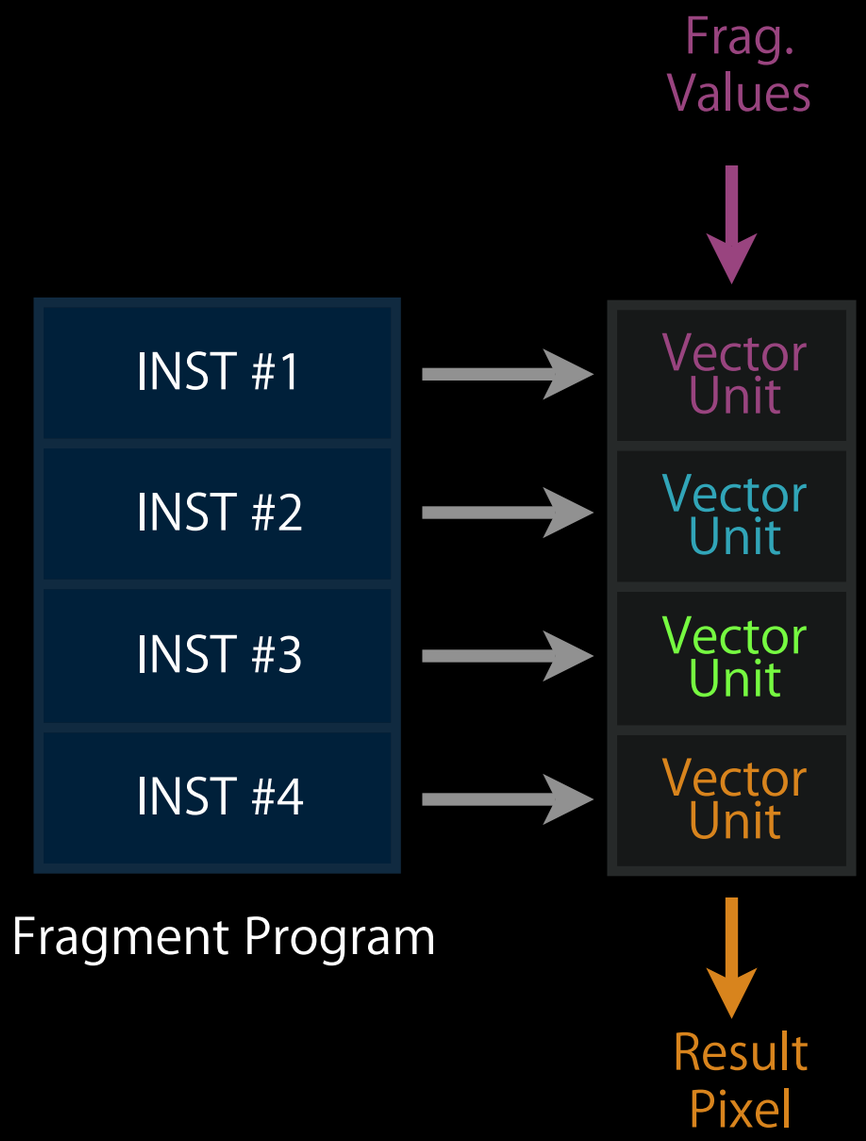
Parallel Processing

Parallelism in “time” allows pipelined execution



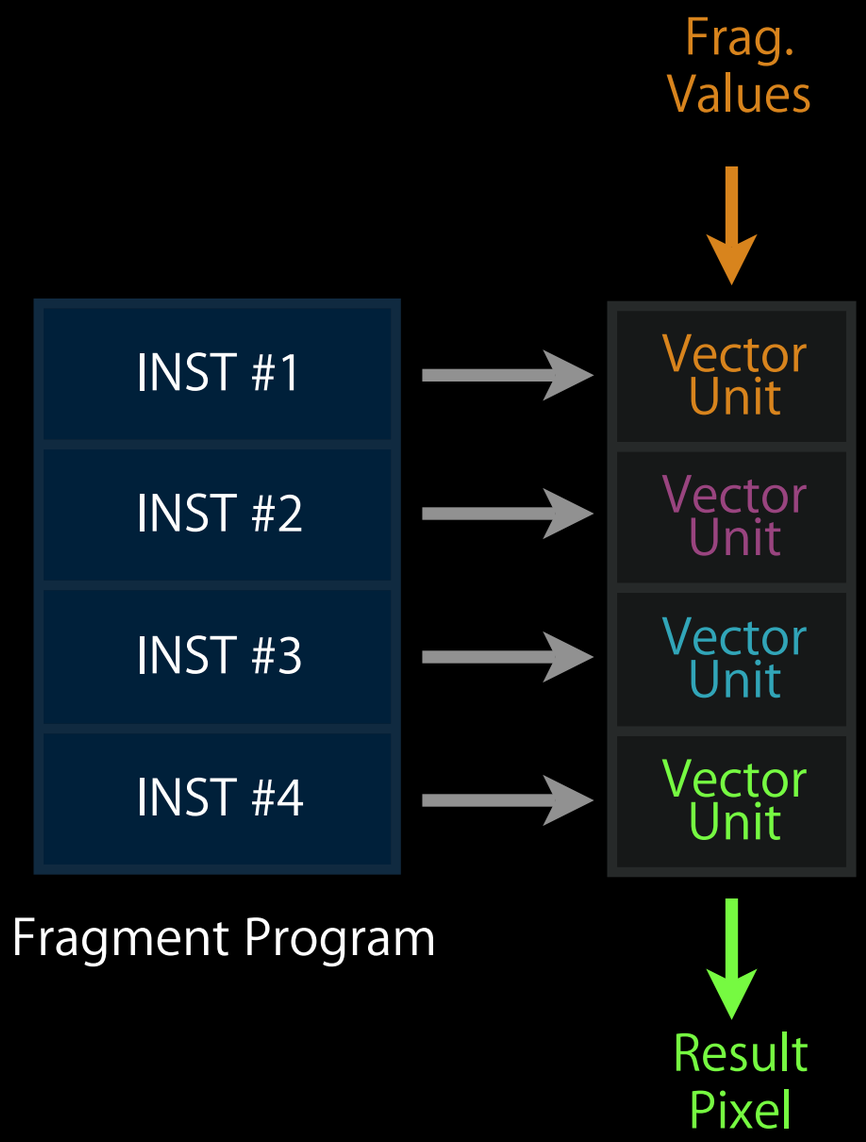
Parallel Processing

Parallelism in “time” allows pipelined execution



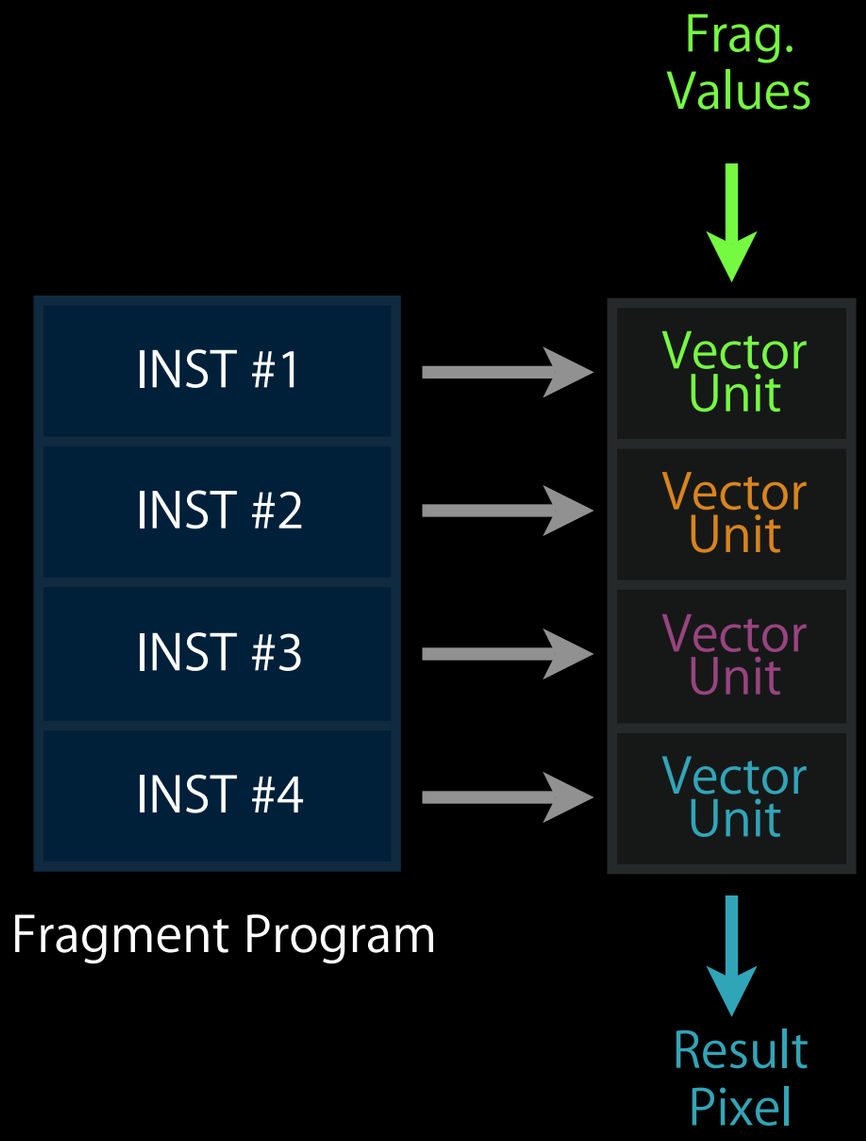
Parallel Processing

Parallelism in “time” allows pipelined execution



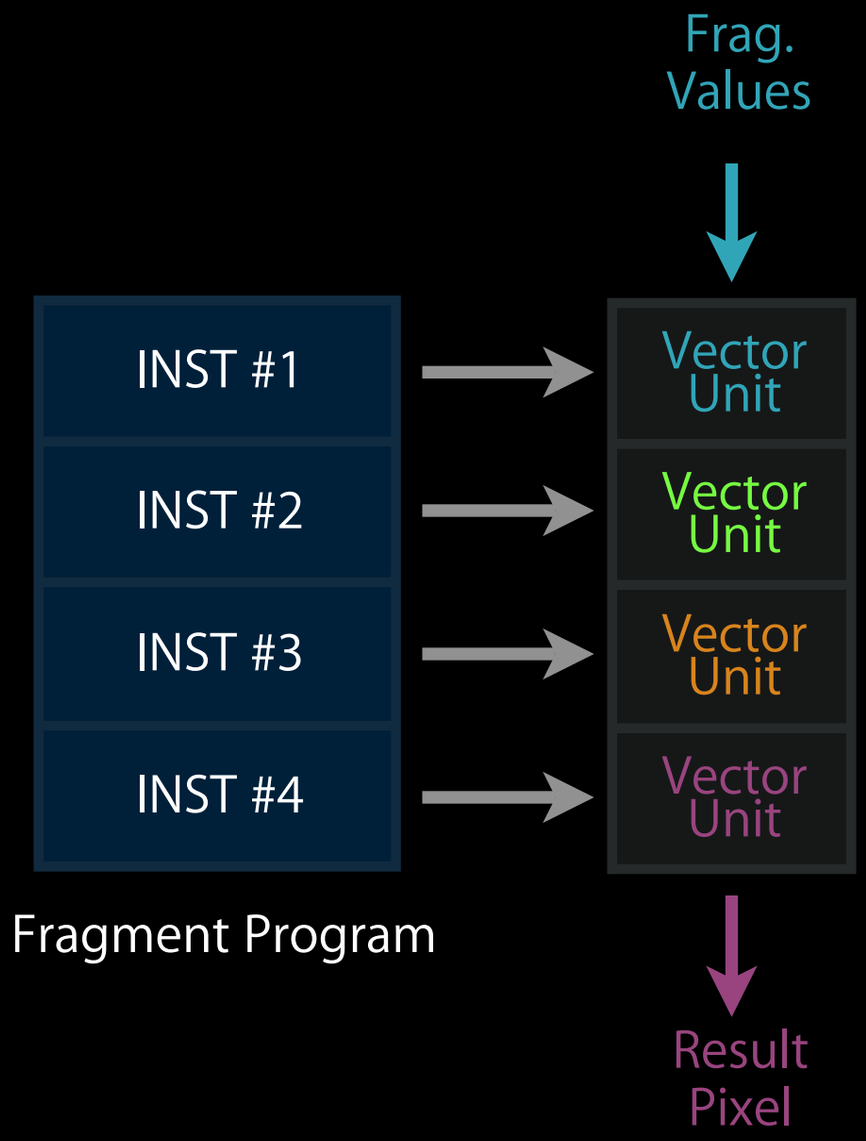
Parallel Processing

Parallelism in “time” allows pipelined execution



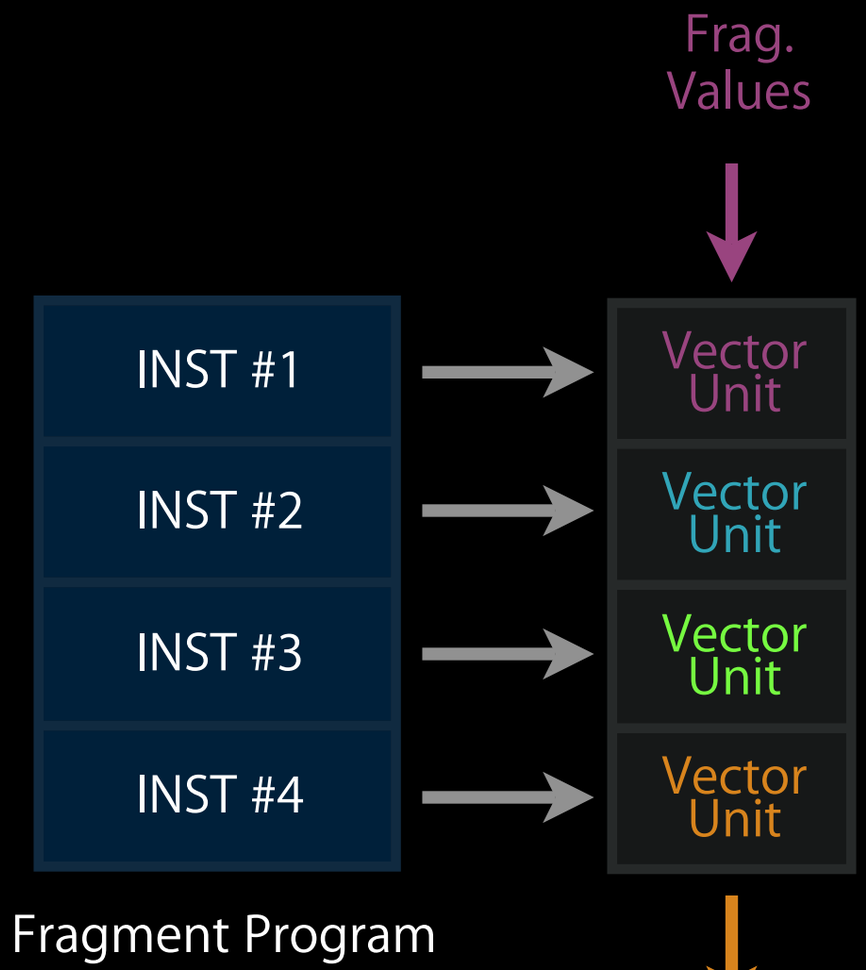
Parallel Processing

Parallelism in “time” allows pipelined execution



Parallel Processing

Parallelism in “time” allows pipelined execution



4 vector operations per clock

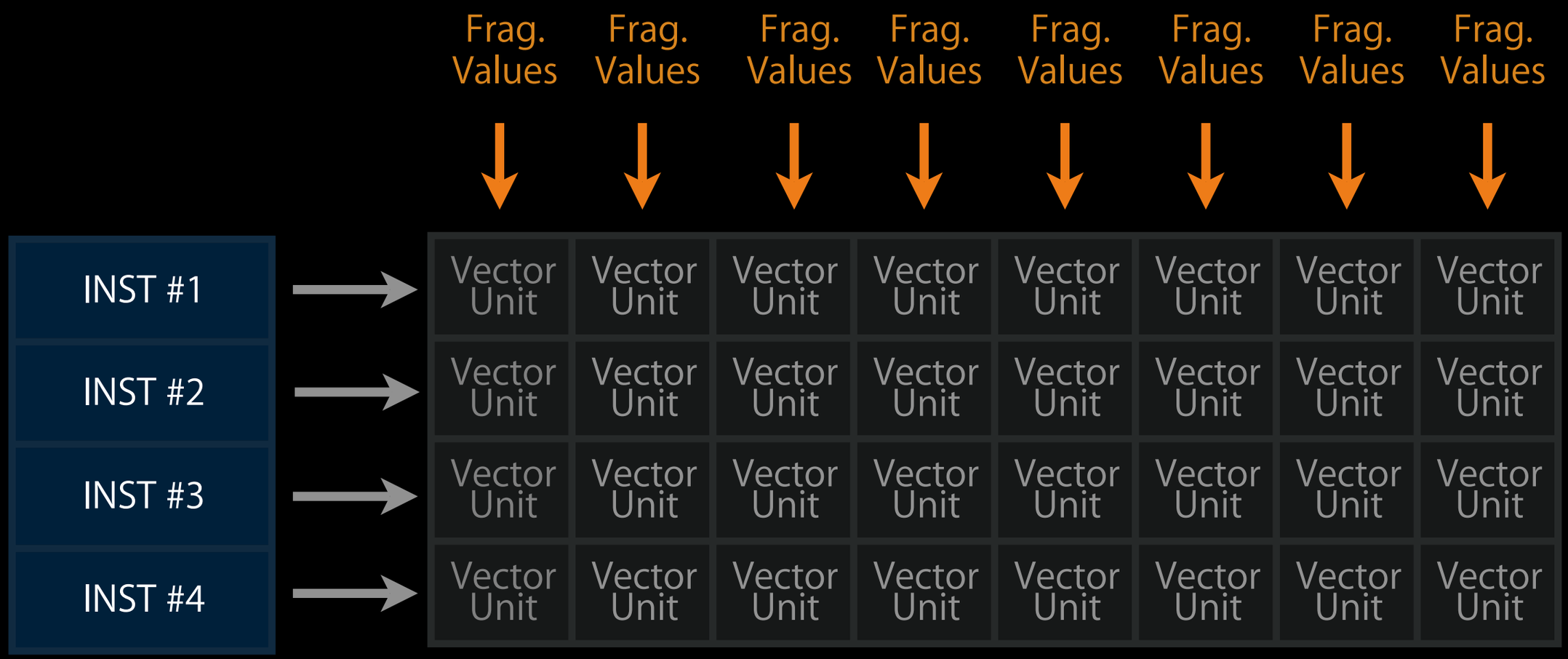
1 result every clock

Putting It Together

- Instruction level parallelism and data parallelism are independent
- Can exploit both at the same time

Parallel Processing

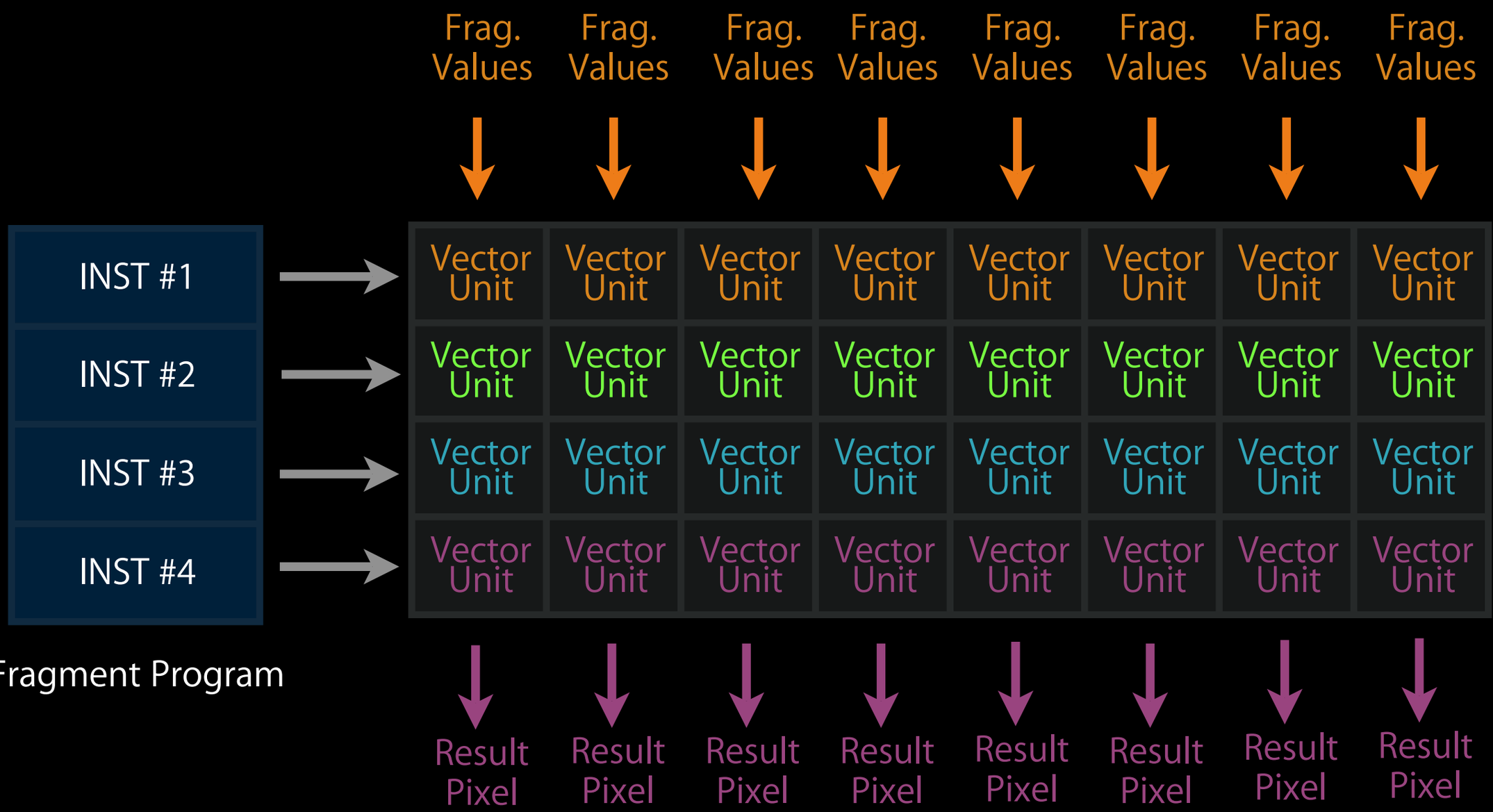
Parallelism allows wide, pipelined execution



Fragment Program

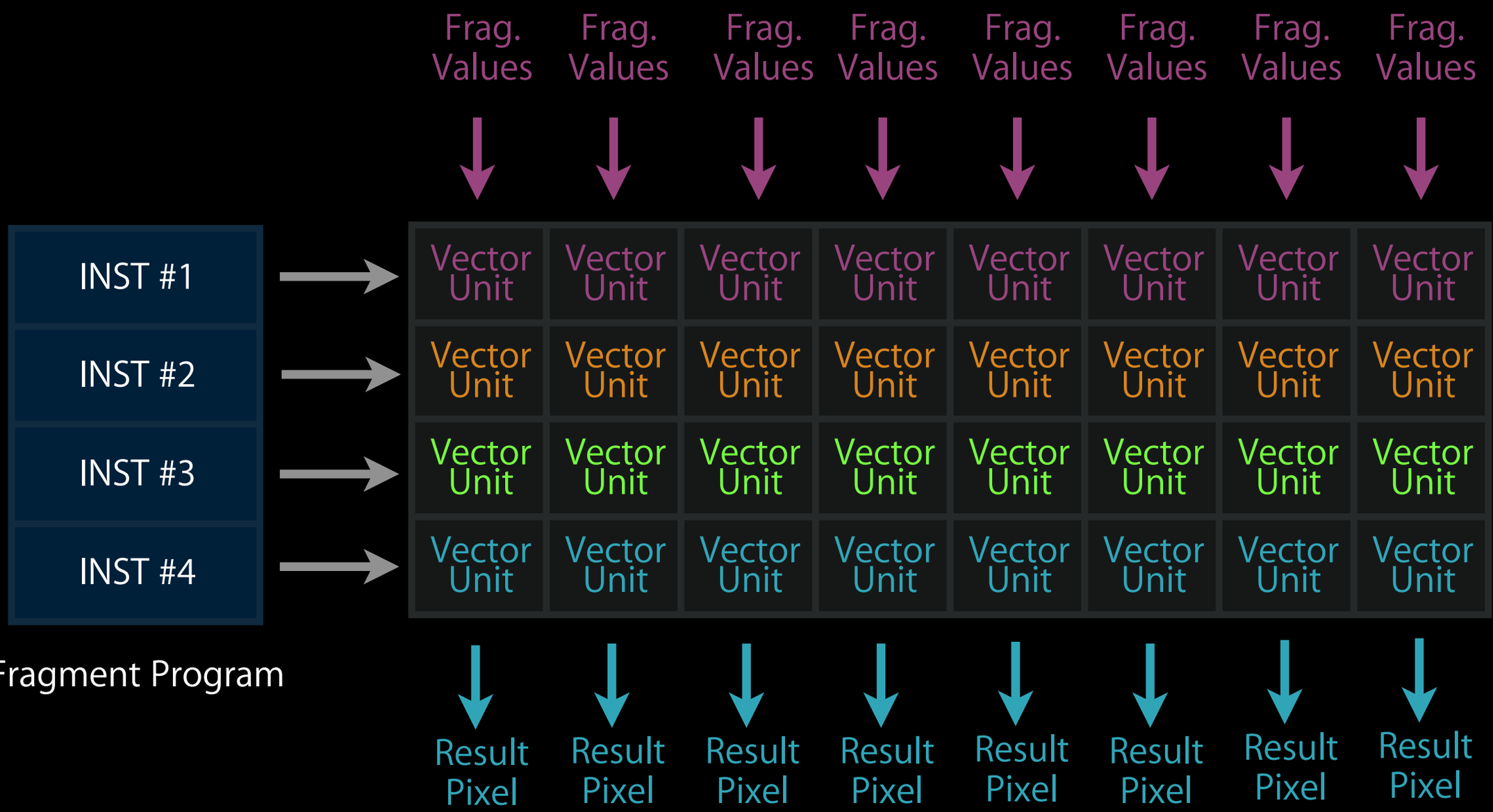
Parallel Processing

Parallelism allows wide, pipelined execution



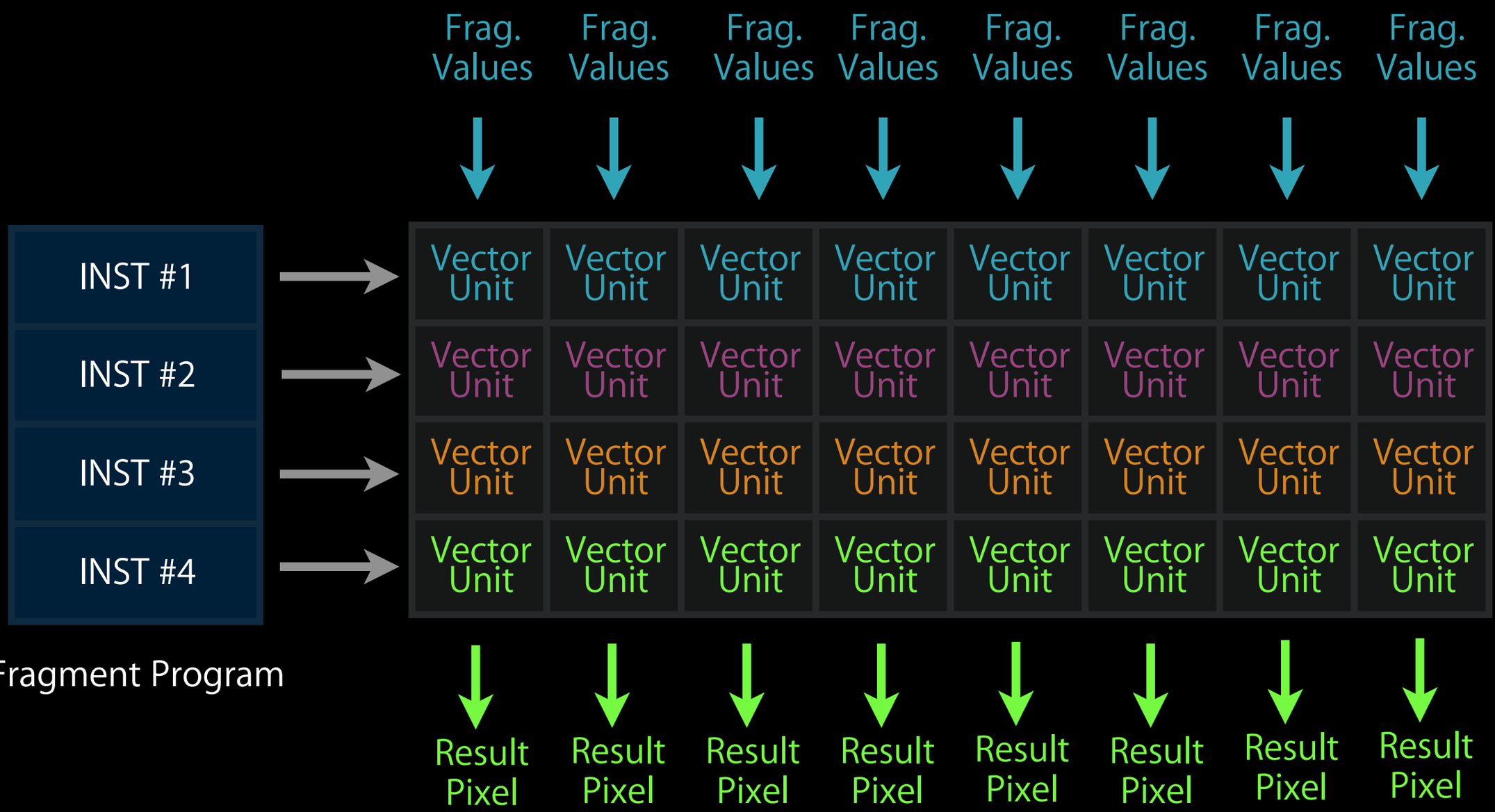
Parallel Processing

Parallelism allows wide, pipelined execution



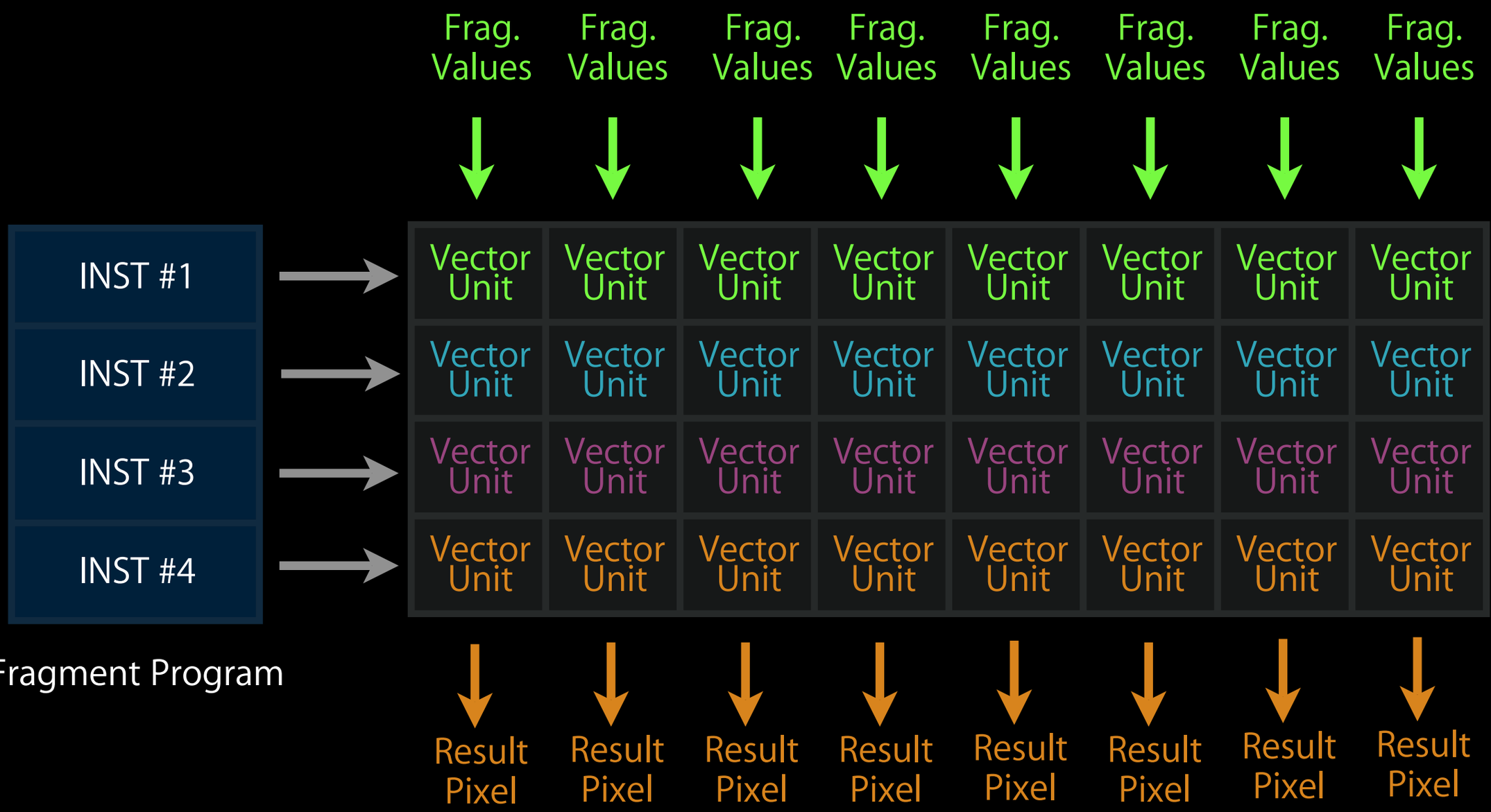
Parallel Processing

Parallelism allows wide, pipelined execution



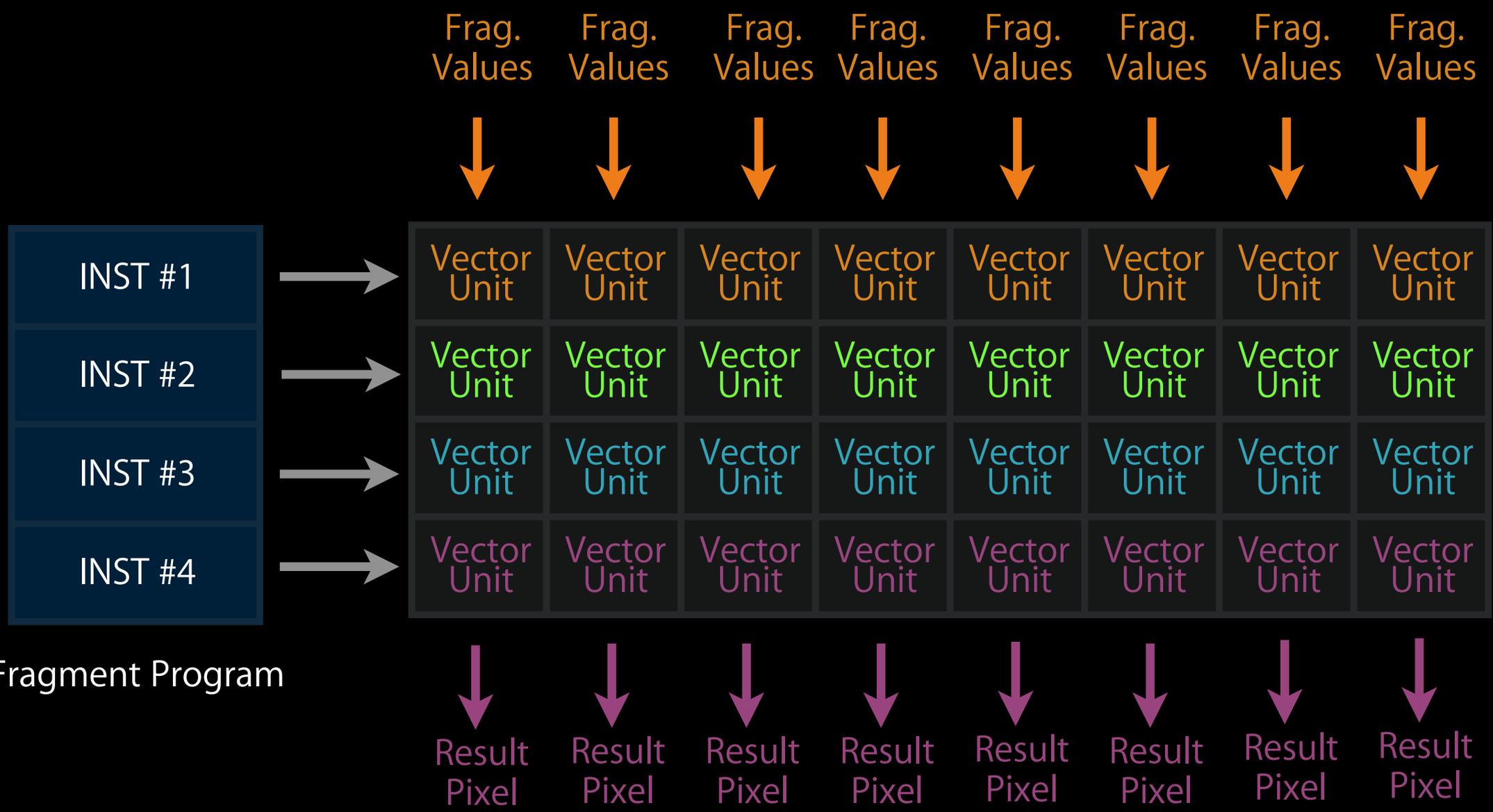
Parallel Processing

Parallelism allows wide, pipelined execution



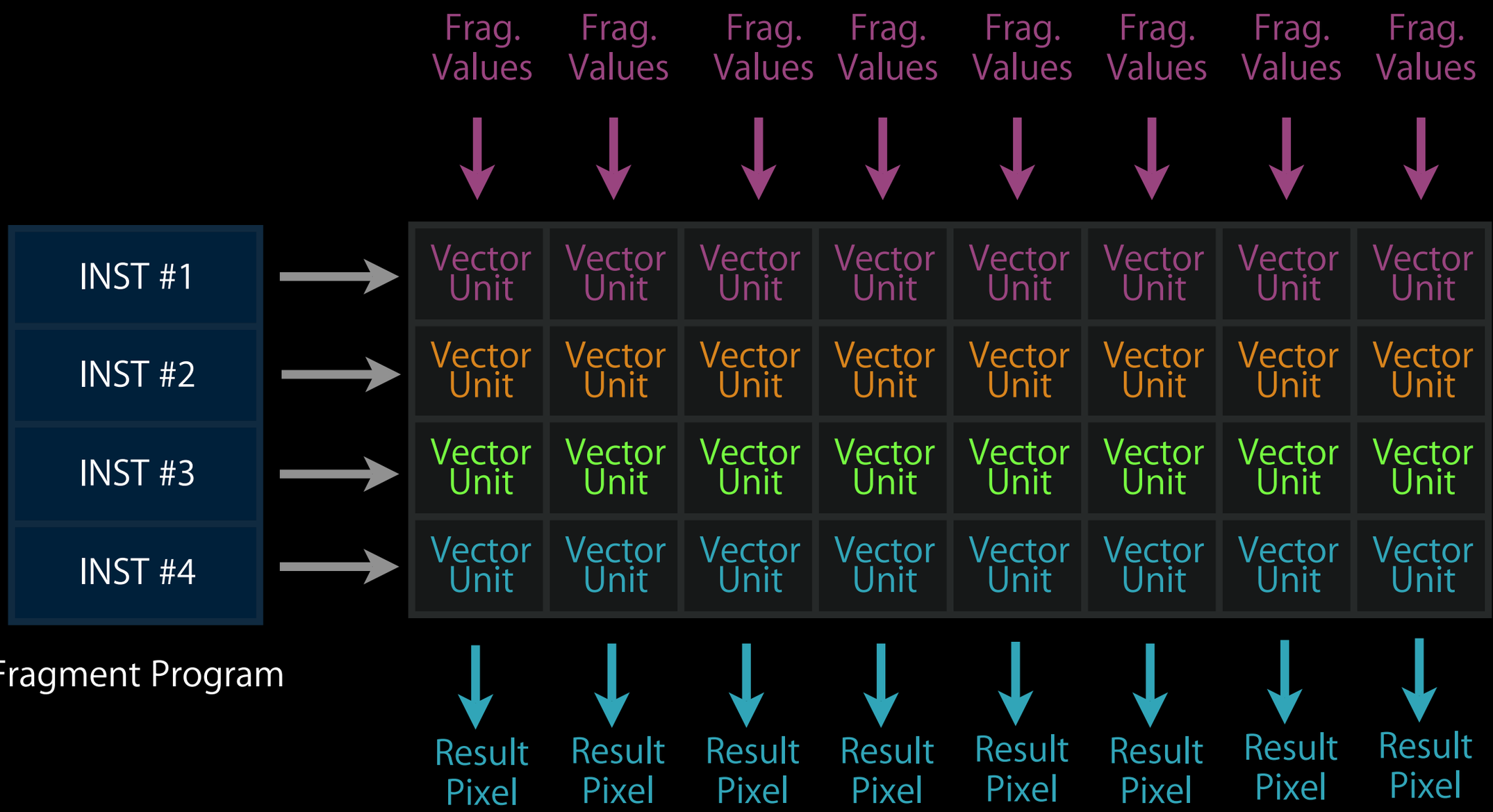
Parallel Processing

Parallelism allows wide, pipelined execution



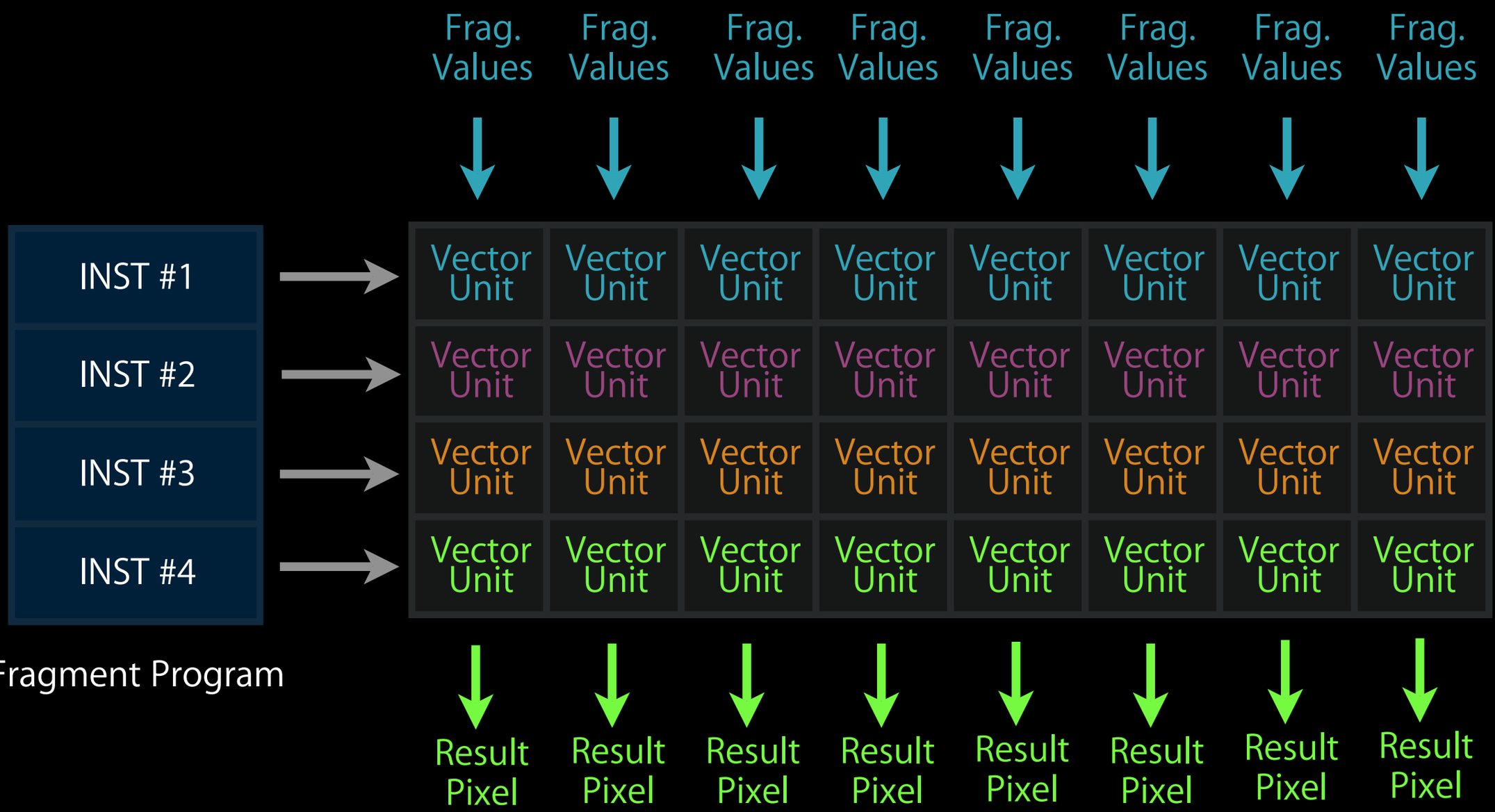
Parallel Processing

Parallelism allows wide, pipelined execution



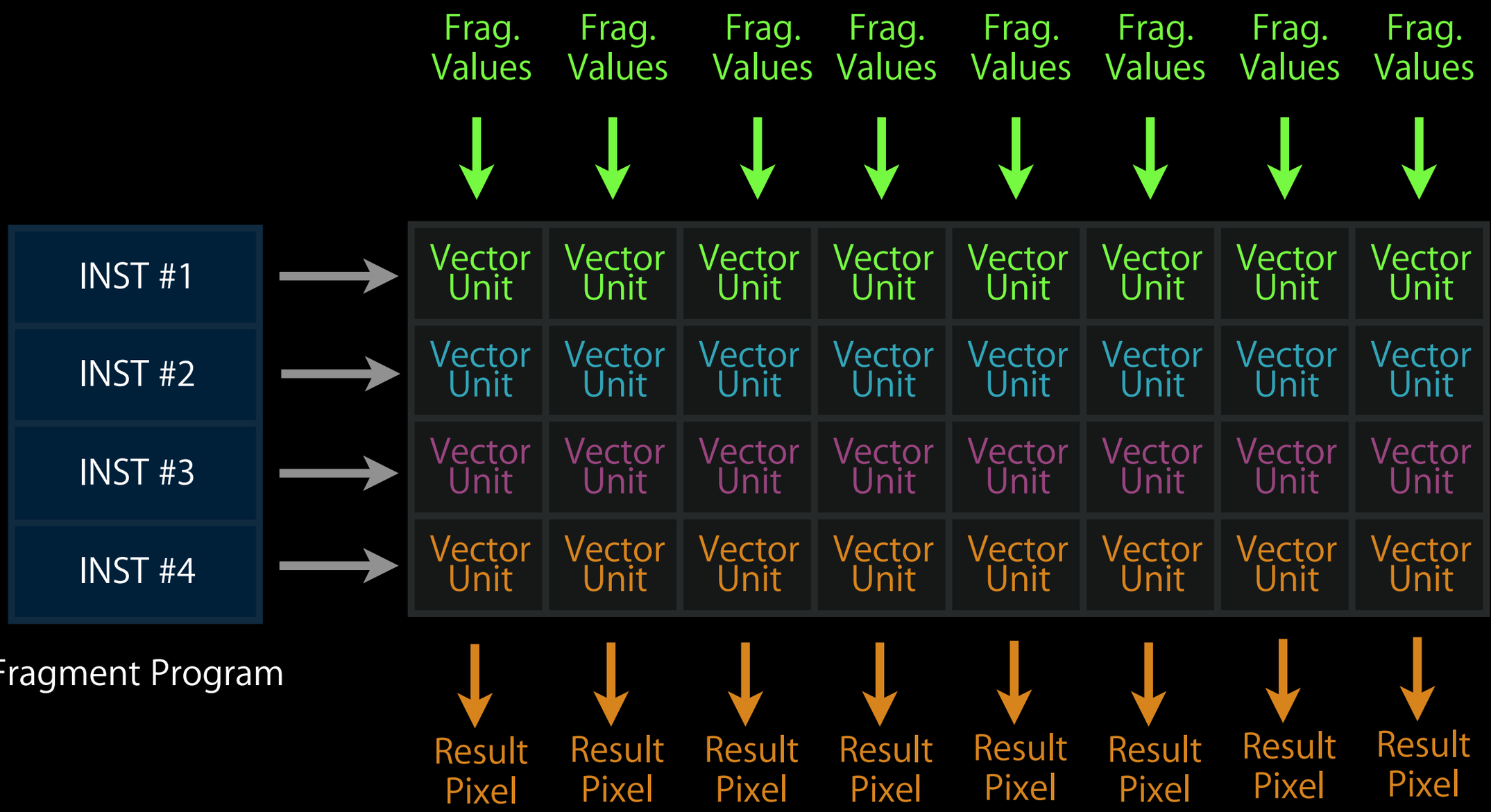
Parallel Processing

Parallelism allows wide, pipelined execution



Parallel Processing

Parallelism allows wide, pipelined execution



Parallel Processing

Parallelism allows wide, pipelined execution

Fragment Program

32 vector operations per clock
8 results every clock

Will It Sustain?

- Chip capability has not peaked
- Parallel Computing possibilities have not peaked
  - 8x4 in space-time has lots of headroom
- Entertainment demand has (probably) not peaked
- Pour the gasoline on!
  - Operating Systems and applications are adding fuel to the fire!

How Do I Program the GPU?

- Use OpenGL
- Standard cross-vendor programming languages
  - `ARB_vertex_program`
  - `ARB_fragment_program`
- Eventually higher level languages…
- … But “assembly-level” programming will always be available

Demo

Fun with Fragment Programming

Summary

- GPUs have an advantage over CPUs for certain classes of data-parallel algorithms
- This advantage is sustainable for the foreseeable future
- You can access this power via OpenGL
- Learn how you can use this stuff before your competitors do :-)

Quartz 2D on OpenGL

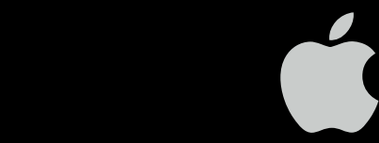
Quartz 2D on OpenGL

- Accelerates Quartz 2D with OpenGL
- Logical next step after Quartz Extreme
- Ties together 2 key thrusts of Mac OS X graphics
- Panther will have initial implementation
- Targeted mainly at OpenGL developers wanting high performance 2D on a OpenGL context

Quartz 2D on OpenGL

```
CGContextRef CGGLContextCreate(CGLContextObj glContext,
          CGSize size, CGColorSpaceRef colorspace);
```

- High-quality 2D rendering
- Virtually identical to software quality
- 2–10x faster than Quartz 2D software rendering
- Quartz quality text cached on textures
- For maxiumum performance, reuse objects
  - CGImageRefs, CGPatternRefs, CGColorRefs
  - Implementation will cache these in video memory



Quartz 2D on OpenGL Limitations

- Certain Quartz 2D operations are not yet supported
  - LCD Text (requires fragment programming)
  - PDF 1.4 blend modes
- CoreGraphics API only
  - No QuickDraw
  - No high-level frameworks (NSImage, etc.)
- Quartz Extreme capable hardware required
- Apps must revisit assumptions about cheapness of accessing window buffer pixels

A Wide Pipe Is Great, But Reading Back Pixels Is Even More Expensive

Vs.

Demo

Quartz 2D on OpenGL

Recap

- New era of innovation in platform graphics
- Mac OS X is a state-of-the-art visual computing platform
- Apple is leading the industry
- Leverage key infrastructure of Quartz 2D and OpenGL
- Learn a few new tricks!

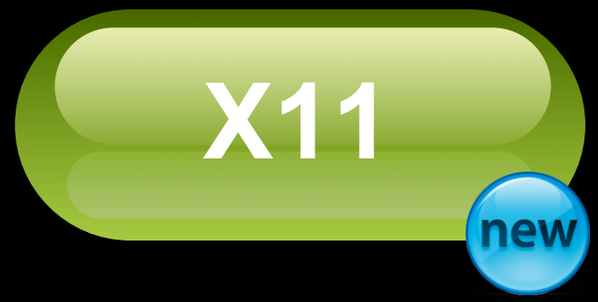
New for Panther

- PDF 1.4 support
- PDF introspection API
- Quartz Scripting—Python Bindings
- CMYK rendering context
- Numerous performance optimizations
  - Text rendering, gradients, printing
- Quartz 2D on OpenGL

New for Panther

207: Quartz 2D In Depth	Presidio Thurs, 9:00am
211: Intro to Quartz Services	Pacific Heights Fri., 9:00am

New for Panther



- XFree86 4.3 Merge
- Double-clickable X11 applications
- Full-screen mode
- Bug fixes

New for Panther

- PostScript support
  - Enables printer sharing with other OSs
- User interface improvements
- Job submission APIs
- CUPS 1.1.19
- CUPs drivers are 1st class citizens
  - gimp-print drivers are in Panther

New for Panther

210: Mac OS X Printing Update

Mission
Thurs., 3:30pm

New for Panther

- End-to-end Color Management by default
  - Cameras embed default profile
  - QTGraphicsImporter applies profile by default
  - CUPS & gimp-print drivers integrated with ColorSync
- Quartz PDF Filters
- Sips—command-line image processing tool
- New APIs for abstract profile generation
- New display calibrator

New for Panther

206: ColorSync Update

Mission
Wed, 5:00pm

New for Panther

- New ways to integrate your app with ImageCapture
  - Automatic Task
  - ImageCapture Services
  - Common UI Layer
- Network support for Image Capture devices
  - Device Sharing and Monitoring

New for Panther

204: Image Capture Update

Marina
Wed., 9:00am

New for Panther

- Fragment programming
- PixelBuffer support
- glCopyTexImage, glTexSubImage optimizations
- Recoverable GPU support
- OpenGLShaderBuilder enhancements
- OpenGLProfiler enhancements
- Use Quartz 2D in a OpenGL context
- Lots of bug fixes and optimizations

New for Panther

201: Mac OS X OpenGL In Depth	Nob Hill Tues., 2:00pm
205: Vertex Programming With OpenGL	Marina Wed., 3:30pm
208: Fragment Programing With OpenGL	Pacific Heights Thurs., 10:30am
209: OpenGL Optimizations	Marina Thurs., 2:00pm
212: Cutting Edge OpenGL Techniques	Russian Hill Fri., 2:00pm

Special Session:
Technology Magic of Keynote

Nob Hill, Tuesday, 3:30pm

Feedback

FF018: Graphics and Imaging Feedback Forum

North Beach
Friday, 5:00pm

WWDC 2003

Worldwide Developers Conference